2001

Debris avoidance strategies for spacecraft

Aries D. Broadnax
San Jose State University

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

DEBRIS AVOIDANCE STRATEGIES FOR SPACECRAFT

A Thesis

Presented to

The Faculty of the Department of Mechanical and Aerospace Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

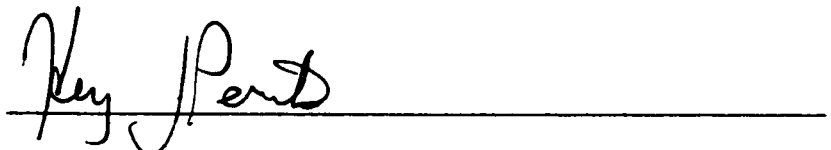
Aries D. Broadnax

August 2001

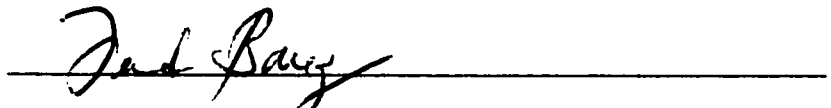
UMI Number: 1405494

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

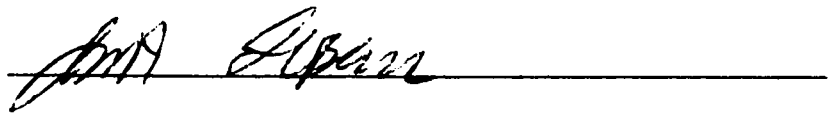
APPROVED FOR THE DEPARTMENT OF
MECHANICAL AND AEROSPACE ENGINEERING



Dr. Henry J. Pernicka, Thesis Advisor

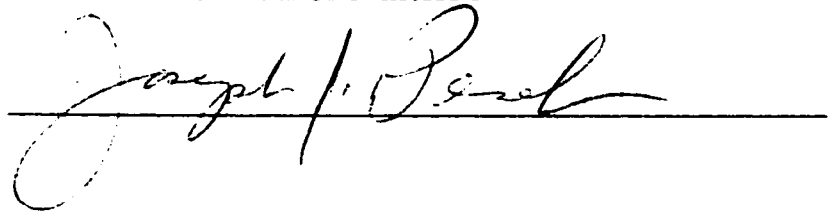


Dr. Fred Barez, Committee Member



Joseph A. LeBlanc, Globalstar Limited Partners, Committee Member

APPROVED FOR THE UNIVERSITY



ABSTRACT

DEBRIS AVOIDANCE STRATAGIES FOR SPACECRAFT

by Aries D. Broadnax

The increased use of outer space has heightened the awareness and concern of the hazards of man-made objects in space. Objects in space that do not serve any purpose are generally referred to as orbital debris. Orbital debris can pose a great threat to operational spacecraft and human space flight. During a spacecraft's lifetime an orbit debris avoidance maneuver may be required to preserve mission life. The scope of this thesis is to create a procedure to determine a debris avoidance strategy that is compatible with a spacecraft's mission objectives.

Using orbital mechanics and optimization techniques a procedure was developed that generates and analyzes many debris avoidance maneuver options. The procedure successfully calculates minimum fuel avoidance maneuvers as well as other maneuver options. A mission analysis team can use this procedure to choose a debris avoidance maneuver that will be compatible with the mission objectives.

Acknowledgements

Special thanks to Brenna, Maya, Jasmine, and Phyllis for their help and patience, XOXO

and to

Dr. Henry J. Pernicka for his guidance and wisdom throughout my undergraduate and graduate studies at San José State University,

and to

Dr. Fred Barez and Joseph A. LeBlanc for becoming my graduate committee members.

Thank you all.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

According to the United States Space Command[1] (USSPACECOM) as of 21 June 2000, there were 2671 man-made satellites in Earth orbit. Earth orbiting satellites are typically used for communications, weather, scientific research, and military applications. These satellites contribute to the total number of objects in Earth orbit. "Over 9,000 objects larger than 10 cm are known to exist. The estimated population of particles between 1 and 10 cm in diameter is greater than 100, 000. The number of particles smaller than 1 cm probably exceeds tens of millions."[2] The increased use of outer space has heightened the awareness and concern of the hazards of man-made objects in space. Figure (1) (observed from some point outside of Geosynchronous Orbit) indicates the large number of objects in Geosynchronous Orbit (GEO) and Figure (2) indicates the large number of objects in Low Earth Orbit (LEO). Objects in orbit that do not serve any purpose are generally referred to as orbital debris.

Figure 1. Objects in Geosynchronous Orbit.[3]

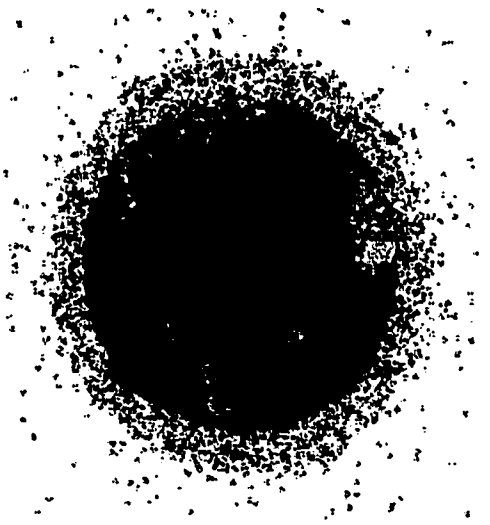


Figure 2. Objects in Low Earth Orbit.[3]

1.1 Sources of Orbital Debris

During the launch of a spacecraft, adapters, boosters or upper stages may be left in orbit. These items are large and can pose a great threat to orbiting spacecraft. Shortly after launch, when solar arrays and booms are deployed, separation bolts and other small objects can be ejected from the spacecraft. The small objects created from these post launch activities have great potential for destruction because they are hard to detect and track. These small particles moving at hyper velocities could severely damage a spacecraft on impact.

When a spacecraft can no longer meet its mission objectives it may be abandoned. An abandoned spacecraft could possibly create debris from explosion or collision with another object. If an explosion or collision occurred, debris of varying shapes and sizes will be created. The Center for Orbital and Reentry Debris Studies (CORDS)[4] states that more than 124 breakups have been verified. Breakups are generally caused by explosions and collisions. However, the cause of all verified breakups is unknown.

The extreme conditions of outer space can cause materials used on spacecrafts, such as paint and insulation, to degrade and separate from the spacecraft producing hazardous particles. Chobotov[5] describes an incident in which a 0.2 mm flake of paint formed a 4 mm crater in the windshield of the Space Shuttle Orbiter STS-07. If objects of near microscopic size can cause damage, the destructive potential of larger objects is easily realized.

These are only some of the sources of orbital debris. As the number of spacecraft in orbit increases the sources of orbital debris and the risk of debris related hazards will

increase.

1.2 Current Solutions

Some ideas for controlling debris include debris collection, spacecraft design considerations, and mission design considerations. Orbital debris collection sounds attractive, however, it could be very dangerous, time consuming, expensive and could potentially create more debris. Designing spacecraft so that objects are not ejected into space during deployment activities will help reduce risk. When the mission lifetime is over or a spacecraft is no longer capable of performing its design tasks, spacecraft re-entry should be considered. For higher orbits, boosting the spacecraft into a "graveyard" orbit might be more reasonable. For re-entry or graveyard orbits, fuel considerations must be made at the design level. Currently, limiting the creation of orbital debris seems to be the best way to reduce risk.

For a spacecraft in orbit, one way to avoid debris (larger tracked pieces) is to perform one or more avoidance maneuvers. During a spacecraft's lifetime an orbit debris avoidance maneuver may be required to preserve mission life. On several occasions the Space Shuttle has performed maneuvers to prevent colliding with other objects. When a debris avoidance maneuver is performed the spacecraft will be removed from its position or "station" on orbit. The mission analysis team must consider whether the spacecraft will return to an orbit that is similar to its original orbit, return to the same orbit at a different position in that orbit, or return to the same orbit and the same position in that orbit, i.e. "return on-station."

1.3 Previous Contributions

The objective of a study by Prussing and Clifton[6] was to determine minimum-fuel, satellite evasive maneuvers. Primer vector theory was used to determine the minimum fuel solutions. The primer vector is the adjoint to the velocity vector and was introduced by Lawden[7] in 1963. A study by Lion and Handelsman[8] was used to obtain a better understanding of the primer vector. The paper discusses the necessary conditions for an optimal solution in terms of the primer vector. The primer vector however, is not the only tool used to determine optimal solutions. In general the problem of optimizing becomes one of minimizing a set of variables or a function of variables using a minimization technique.

Applied Optimal Control by Bryson and Ho[9] describes and outlines optimization techniques for many different problems and systems. Understanding how these techniques are applied to different problems provides insight on how to choose a technique that will meet given optimization goals. Mathematics Professor H. J. Greenberg[10], from the University of Colorado at Denver, created an online Mathematical Programming Glossary. This glossary is an alphabetized list of definitions, equations, formulas, outlines, and algorithms specific to mathematical programming. Programming information on certain optimization techniques can be found here.

In the Ph.D. thesis by D'Amario[11], the application of primer vector theory and optimization in calculating minimum impulse trajectories is described. The dissertation has an overall step-by-step summary of the trajectory optimization process. This dissertation along with the papers and studies mentioned above provides the necessary

foundation to apply primer vector theory and optimization techniques to the current study.

The Prussing and Clifton[6] study focused on the return-on-station option mentioned above. In general the return on-station maneuvers were multiple-impulse maneuvers that moved the spacecraft to a safe orbit then rendezvoused with the original orbit station. The minimum allowable separation distance between the spacecraft and the debris determines the safe orbit. In the concluding remarks of the paper, Prussing and Clifton[6] state that "The return on-station maneuvers are probably the most expensive in terms of fuel required, compared with other less-exacting maneuvers, but when optimized, they provide benchmarks with which cost of other maneuvers can be compared." This statement helps define the scope and objectives of this thesis.

2.0 Current Study

The scope of this study is to create a procedure to determine a debris avoidance strategy that is compatible with a spacecraft's mission objectives. This procedure includes: the recognition of possible threat conditions; the definition of a minimum safe distance; the verification that a debris avoidance maneuver is required and the evaluation of maneuver possibilities. The possible courses of action are organized in a flow chart to clarify the proposed debris avoidance procedure. Figure (3) shows the main section of the flow chart.

Most spacecraft and orbital debris are located in LEO. To create a practical scenario a circular, equatorial, LEO, with an orbital period of approximately ninety minutes was used for this study. The debris' orbit is also a ninety minute LEO, however

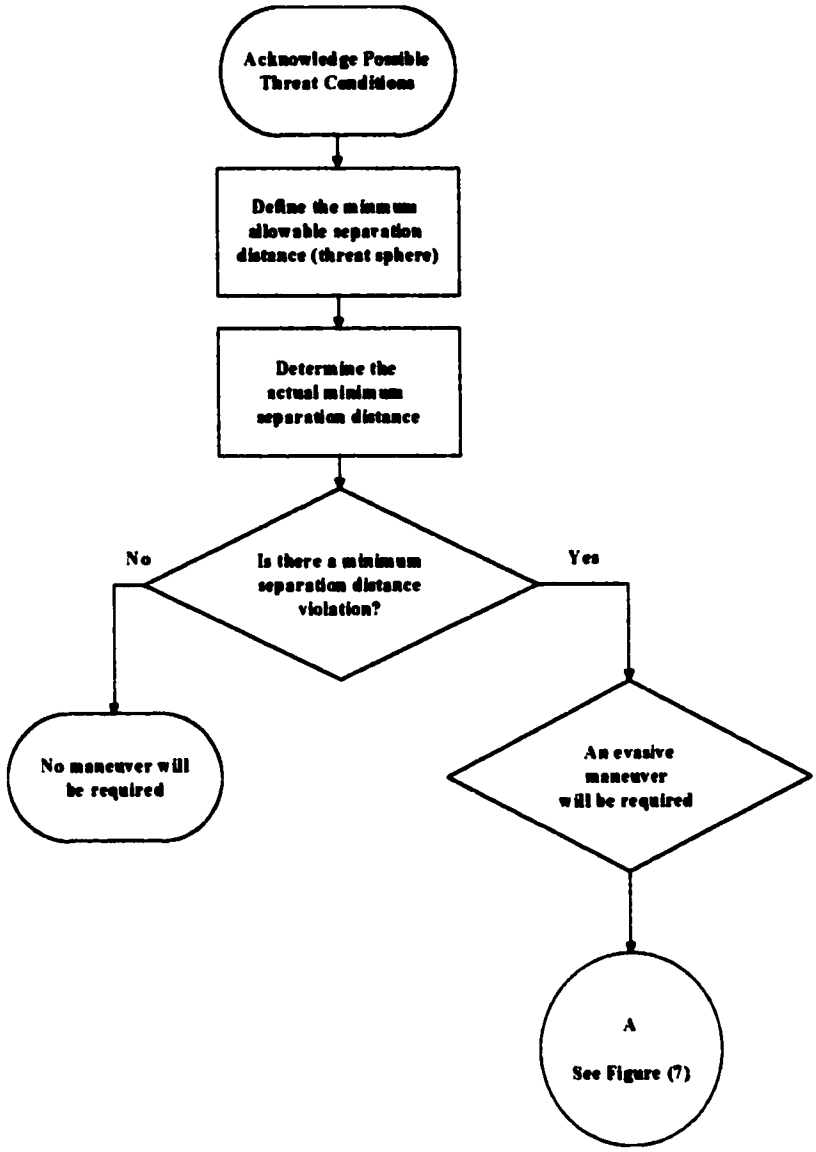


Figure (3) Debris Avoidance Procedure Flow Chart (Main Section).

it is slightly elliptical and inclined ninety degrees to the equator in a polar orbit. This scenario represents a spacecraft in imminent danger that must perform an orbital debris avoidance maneuver in less than one orbital period.

A licensed copy of MATLAB® Student Version 5.3.0.14912a Release 11 from The Math Works Inc. along with an Orbital Mechanics MATLAB® Tool Box from

Science Software was used for programming purposes to produce the results shown in this thesis.

3.0 The Debris Avoidance Procedure

The following sections describe the main and maneuver selection portions of the debris avoidance procedure flow chart.

3.1 Acknowledge Threat Conditions

The first step in the debris avoidance process is to recognize that a threat may exist. Figure (4) below shows an example of a possible threat condition. The close approach information may come from a debris tracking organization such as the USSPACECOM[1], an organization responsible for cataloging and tracking man-made objects in space, or from other institutions that have the capabilities to determine risk. The information may come in the form of a debris state vector along with the close approach date and time. If the close approach involves another operational spacecraft the avoidance process may have to be coordinated with other parties.

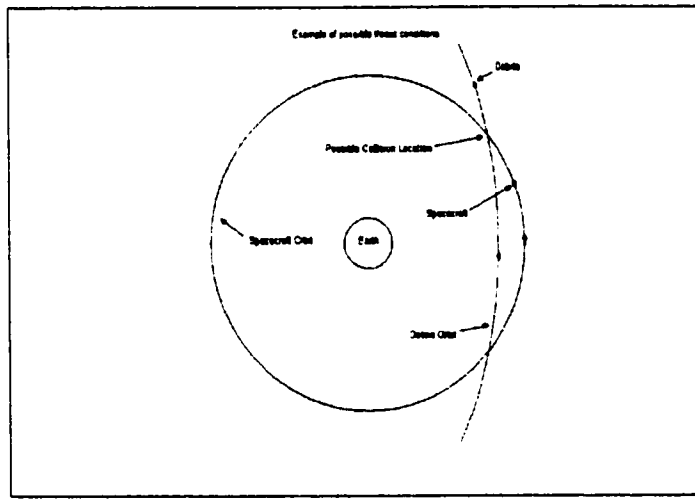


Figure (4) Possible Threat Condition.

3.2 Determine Threat Sphere Size

When a close approach is imminent the question of "How close is too close?" must be answered. Typically a threat sphere or box around the spacecraft is defined. There is no formula to determine the size of a threat sphere. The mission analysis team must determine a threat sphere geometry that is compatible with the spacecraft's mission requirements and restrictions. Information such as the size of the debris, the size of the spacecraft and its maneuvering capabilities may help in determining a threat sphere size. Figure (5) shows an example of threat sphere geometry.

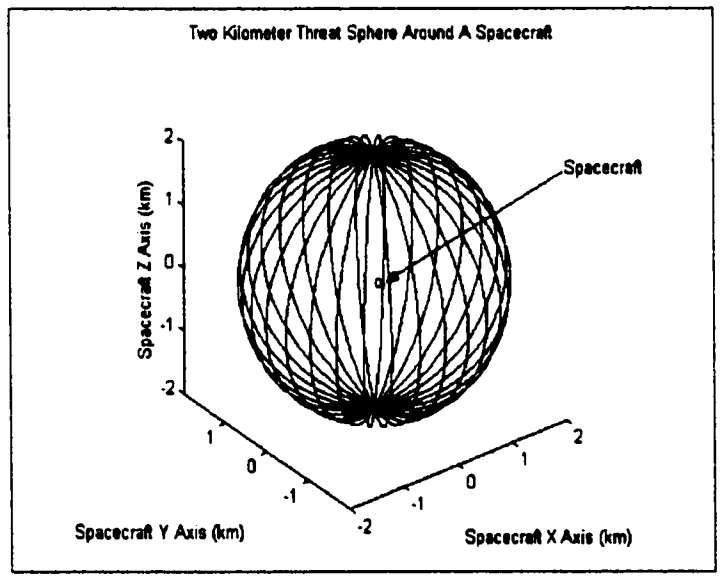


Figure (5) Threat Sphere Geometry.

NASA Flight Rule A4.1.3-6[12], states "A collision avoidance maneuver will be performed for a conjunction predicted by the United States Space Command if the predicted miss distance is less than 2 km radially, 5 km down track, and 2 km out of plane and if the maneuver does not compromise either primary payload or mission

objectives. Propellant redlines will not be budgeted for any potential maneuvers." This flight rule is a good example of how mission objectives can influence a decision to perform a debris avoidance maneuver. A threat sphere of 2 km was arbitrarily used for this study.

3.3 Determine the Actual Minimum Separation Distance

After a threat sphere has been defined a high fidelity orbit propagator should be used to determine the actual minimum separation distance between the spacecraft and the debris. If the minimum separation distance is less than the size of the threat sphere an evasive maneuver would be required. Debris passing within the boundary of a threat sphere is illustrated in Figure (6).

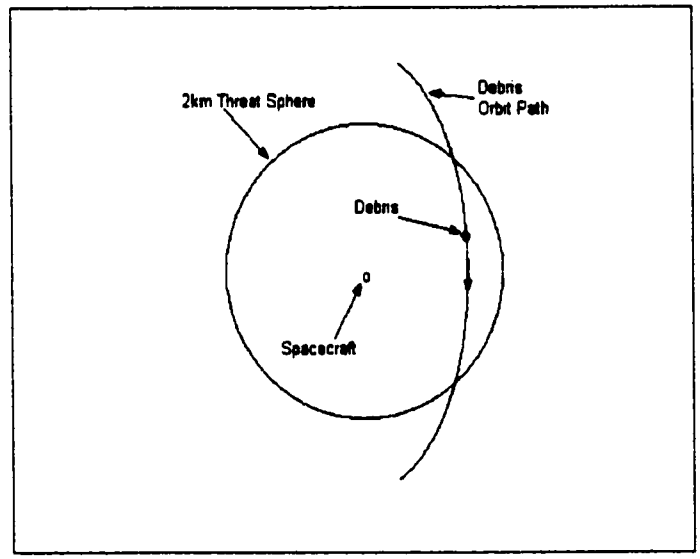


Figure (6) Threat Sphere Violation.

In this study a program was created to determine the actual minimum separation distance between the spacecraft and the orbiting debris. The program reads in the two state and time vectors from an initial conditions file. The state vectors are expressed in

Cartesian Earth Centered Inertial (ECI) coordinates as

$$sv = [x \quad y \quad z \quad \dot{x} \quad \dot{y} \quad \dot{z}] \tag{1}$$

where $r = [x \quad y \quad z]$ is the position vector in kilometers (km), (2)

and $v = [\dot{x} \quad \dot{y} \quad \dot{z}]$ is the velocity vector in kilometers per second (km/s). (3)

The program then prompts the user for a close-approach search duration, and a maneuver threat sphere size that defines the minimum allowable separation distance before a maneuver is required. The state vectors are propagated to a common time that is specified in the initial conditions file. The orbit propagation is done using Cowell's method.

Cowell's method, which is outlined in *Fundamentals of Astrodynamics*[13], involves writing the equations of motion including all perturbations and numerically integrating them. For the problem in question the equation of motion is

$$\ddot{r} + \frac{\mu}{r^3} r = a_p \tag{4}$$

where μ is the Earth's gravitational parameter,

$$r = \sqrt{x^2 + y^2 + z^2} \tag{5}$$

is the magnitude of the position vector, and a_p is the perturbing acceleration, which could be caused by the gravity of the Earth, Moon, Sun, other perturbing bodies or a combination there of. For this study a_p includes Earth's gravity, the Earth's oblate shape (J_2), the Sun, and the Moon. For numerical integration the vector equation of motion must be broken down into scalar form as

$$\dot{x} = v_x \tag{6}$$

$$\dot{y} = v_y \tag{7}$$

$$\dot{z} = v_z \tag{8}$$

$$\dot{v}_x = a_{px} - \frac{\mu}{r^3} x \tag{9}$$

$$\dot{v}_x = a_{py} - \frac{\mu}{r^3} y \tag{10}$$

$$\dot{v}_x = a_{pz} - \frac{\mu}{r^3} z \tag{11}$$

A fourth-order, variable step size, Runge-Kutta numerical integration routine (MATLAB® ode45.m) was used to integrate the equations of motion developed using Cowell's method. With a variable step size and multiple perturbation calculations, ode45.m becomes a CPU time-intensive routine. However, Cowell's method gives an accurate representation of spacecraft motion about the Earth.

A fourth-order, fixed step size, Runge-Kutta numerical integration routine (rk4fixed.m) is used to integrate the equations of motion, independently, using the same step size. This provides a simple means for comparing the distance between the state vectors at each identical time step, provided the integration start time is the same for both state vectors.

The search for the minimum separation distance begins by checking for a violation of the threat sphere. The distance between the two position vectors, d, is evaluated at each integration step along their orbit paths. The distance is calculated as

$$d = \sqrt{(x_{sc} - x_{deb})^2 + (y_{sc} - y_{deb})^2 + (z_{sc} - z_{deb})^2} \tag{12}$$

where the subscripts "sc" and "deb" denotes the spacecraft and debris' position vector

components respectively. As the propagation continues, if and when d becomes slightly larger than the threat sphere size, the step size is reduced and the search for the minimum separation distance continues. Once it is found, the program outputs: the date and time the threat sphere violation begins and ends, the date and time of the actual minimum separation distance, and the state vectors at the time the minimum separation distance occurs. This output information is passed into other programs for the maneuver selection portion of the debris avoidance procedure. For clarity, the following is a list of the steps described above.

1. Read in the spacecraft and orbital debris state vectors. $sv_{sc}(t_{sc})$, $sv_{deb}(t_{deb})$
2. Define the search duration, and the maneuver threat sphere size.
3. Propagate the orbits to a common time. $sv_{sc}(t_0)$, $sv_{deb}(t_0)$ (variable step size routine).
4. Propagate the orbits, calculating d at each step, until d is slightly larger than threat sphere size (fixed step size routine).
5. Propagate the orbits from t_0 to the time (defined by step 4) where d becomes slightly larger than the threat sphere size (variable step size routine).
6. Propagate the orbits with a smaller step size, calculating d at each step, until the minimum separation distance is found (fixed step size routine).

The input to the search program used for this study is:

$$t_{sc} = [1 \quad 1 \quad 2001 \quad 12 \quad 0 \quad 0]$$

$$sv_{sc}(t_{sc}) = [6572.135537 \quad 1026.818761 \quad 0.000760 \quad -1.196824 \quad 7.648001 \quad 0]$$

$$t_{deb} = [1 \quad 1 \quad 2001 \quad 12 \quad 22 \quad 30]$$
$$sv_{deb}(t_{deb}) = [-1156.679935 \quad 0.000365 \quad 6548.222253 \quad -7.617280 \quad 0 \quad -1.343370]$$

The format of the time vector is t = [day month year hours minutes seconds].

The time vector is converted to a scalar Julian date. The Julian date is the number of days since 12:00 U.T. time January 1, 4713 B.C.

The output from the search program for this study is:

Minimum Separation Distance Violation Begins
1.999920e+000 km @ 01-01-2001 13:27:29.8174

Minimum Separation Distance Violation Ends
2.000101e+000 km @ 01-01-2001 13:27:30.1827

Actual Minimum Separation Distance
9.107781e-005 km @ 01-01-2001 13:27:29.9987

$$t_{msd} = [1 \quad 1 \quad 2001 \quad 13 \quad 27 \quad 30]$$
$$sv_{sc}(t_{msd}) = [6652.000004 \quad -0.010263 \quad 0 \quad 0.000012 \quad 7.740923 \quad 0]$$

$$t_{msd} = [1 \quad 1 \quad 2001 \quad 13 \quad 27 \quad 30]$$
$$sv_{deb}(t_{msd}) = [6652.000005 \quad 0 \quad -0.010254 \quad 0.000012 \quad 0 \quad 7.743250]$$

The output indicates that the minimum separation distance is approximately 91 cm. This is clearly a debris avoidance maneuver situation. The output from the search program is then used to determine the maneuver possibilities.

3.4 Maneuver Selection

Once it is established that an evasive maneuver is required, the mission analysis team must decide from which maneuver category to choose. Figure (7) shows the maneuver selection portion of the flow chart.

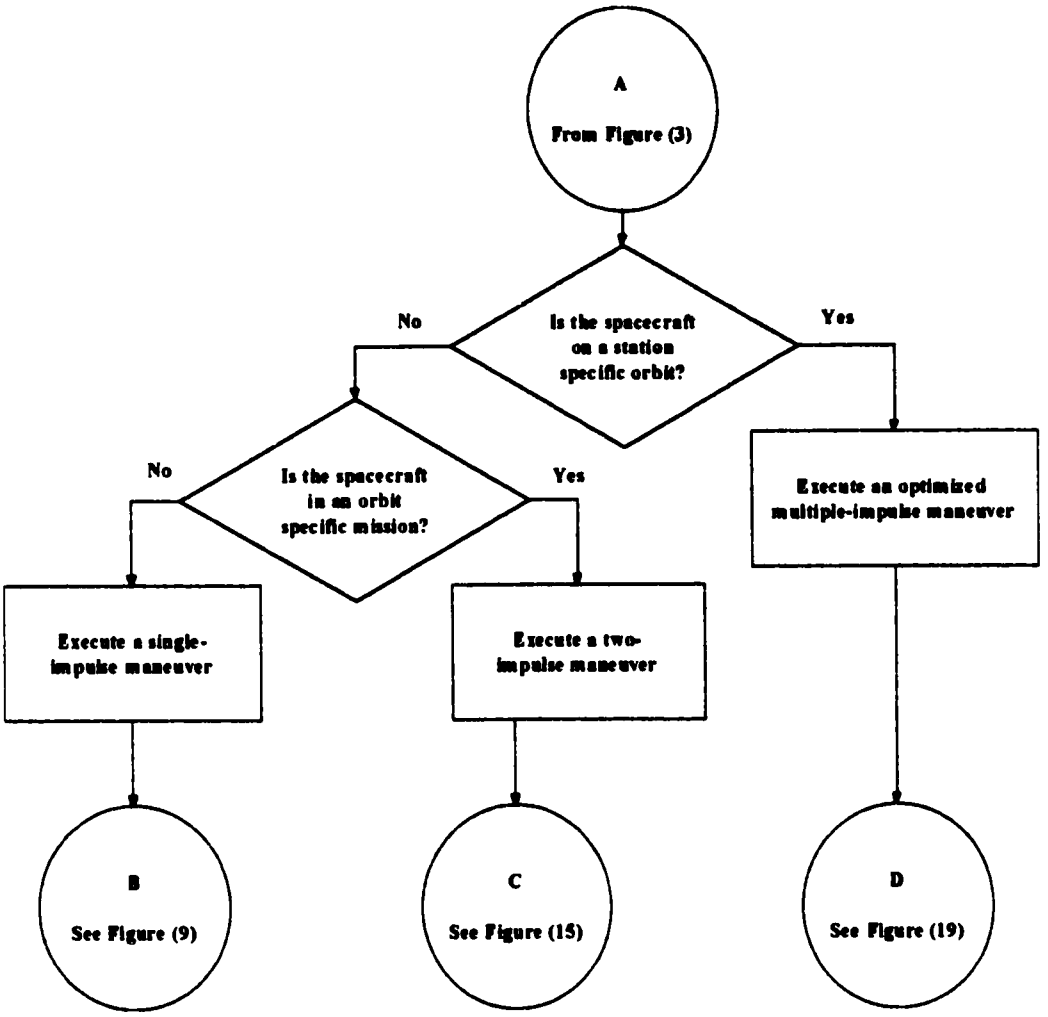


Figure (7) Debris Avoidance Procedure Flow Chart (Maneuver Selection Section).

If the spacecraft is in a station-specific orbit and must return to that station after the evasive maneuver is performed, the optimized multiple-impulse maneuver should be used (flow chart Option D). The first impulse sends the spacecraft on a trajectory that will avoid the debris. The second impulse puts the spacecraft on a trajectory back towards its original station and a third impulse will rendezvous with the station. However, Primer Vector analysis (discussed in detail in the Multiple-Impulse Maneuver section of this thesis) may indicate that an additional impulse will lower the total ΔV.

If the mission is orbit-specific but not station-specific a two-impulse maneuver could be used (flow chart Option C). The first impulse puts the spacecraft on the debris-avoiding trajectory and a second impulse targets the original orbit. If there are time constraints for the orbit-specific case the multiple-impulse maneuver can be used, however, in general, it will not be "optimized."

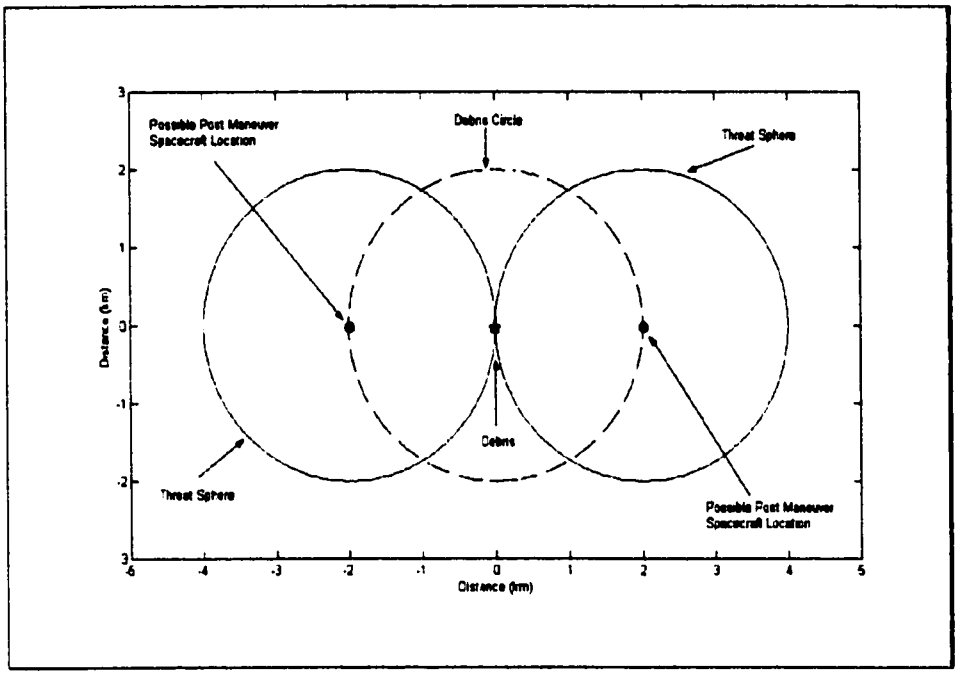


Figure (8) Debris Avoidance Circle.

A single impulse maneuver (flow chart Option B) can be performed if the mission objectives can be met on an orbit that differs slightly from the original. A single impulse maneuver will place the spacecraft in a new orbit that will steer clear of the debris. Steering clear of the debris implies that after the maneuver, the minimum separation distance between the spacecraft and debris will be exactly the same as the threat sphere radius. Avoiding the debris by the exact threat sphere distance, as opposed to a larger avoidance distance, will likely minimize the ΔV required. Figure (8) shows two (of an

infinite number of) possible desired locations of the spacecraft at closest approach to the debris after the avoidance maneuver. A circular locus of possible satellite-debris closest approach locations is thus formed around the debris. This circle is referred to as the "avoidance circle."

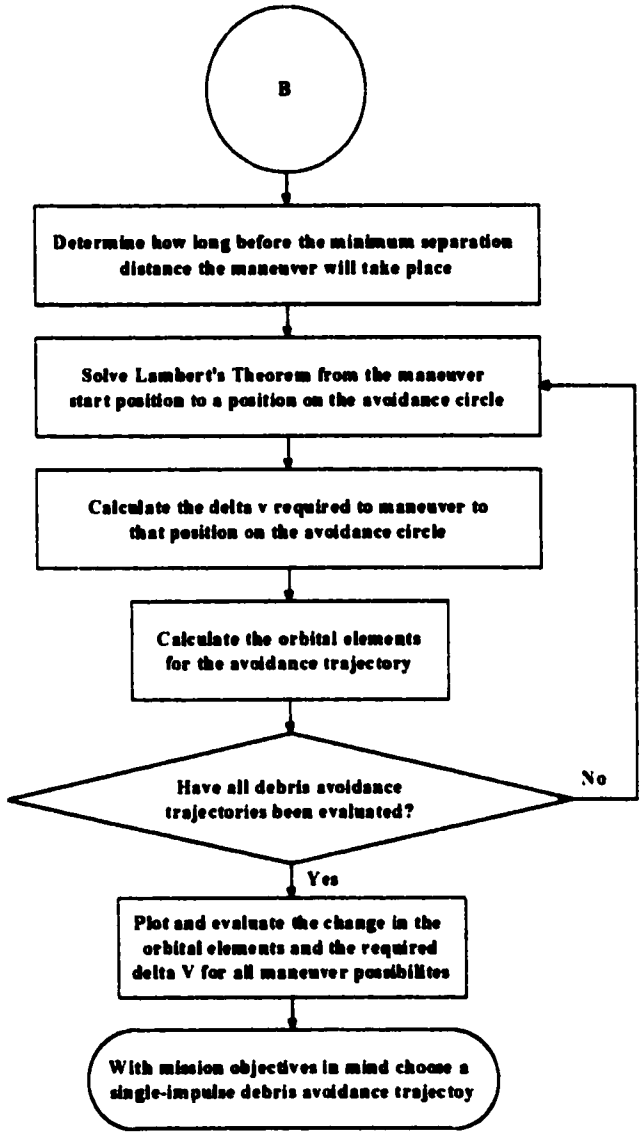


Figure (9) Debris Avoidance Procedure Flow Chart (Single-Impulse Maneuver Section).

4.0 Single-Impulse Maneuver

Figure (9) shows the single-impulse maneuver portion of the debris avoidance flow chart. A program was written to calculate single-impulse evasive maneuvers to multiple locations around the avoidance circle. The input to the program is the number of hours, minutes, and seconds prior to the minimum separation time that the evasive maneuver will take place. The maneuvers are generated to target terminal positions around the avoidance circle at an arbitrarily chosen increment of five degrees. The avoidance circle geometry is shown in Figure (10).

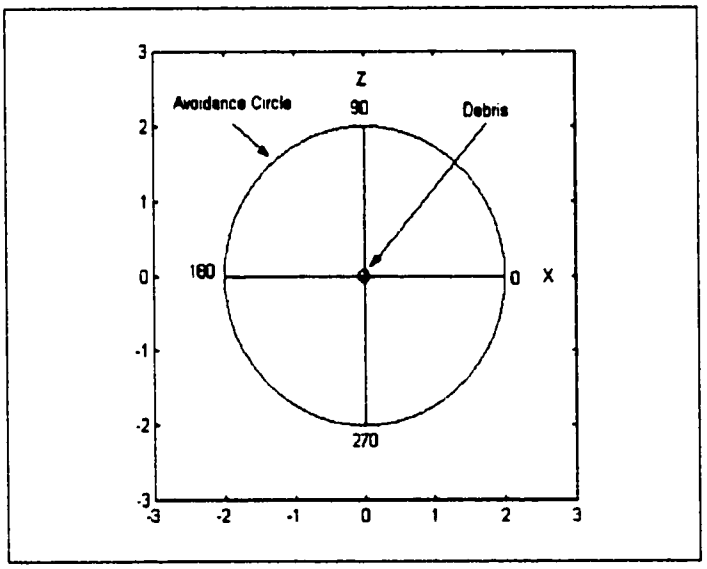


Figure (10) Avoidance Circle Geometry.

The ΔV for the maneuver is calculated by solving Lambert's theorem (orbit determination from two position vectors and the time of flight between them) from the initial spacecraft position to a final position on the avoidance circle. The method used to solve Lambert's theorem is "The p-Iteration Method." This method starts with a trial value of the semi-parameter p. Figure (11) shows the geometry of an elliptical orbit. The semi-parameter is

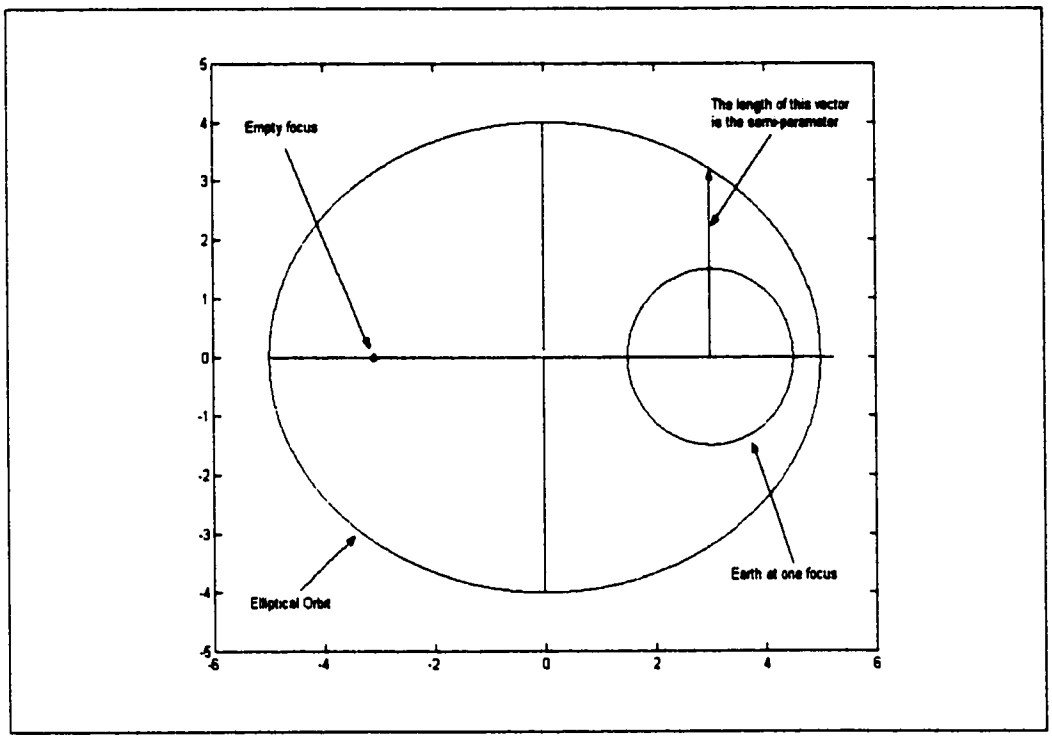


Figure (11) Elliptical Orbit Geometry.

used to determine an orbit that contains both position vectors. The time of flight between the two position vectors is compared to the desired time of flight. The semi-parameter is changed iteratively until the time of flight matches the desired time of flight. A detailed description of the *p*-iteration method can be found in Reference (13). The *p*-Iteration method provides a two-body estimate of the avoidance trajectory initial conditions. A differential corrections algorithm must be applied to find the correct initial conditions. In order to solve Lambert's theorem, the points on the avoidance circle must be expressed in ECI coordinates. Equations (13-15) quantify the change in the ECI position vector to the debris required to obtain an ECI position vector that is located on the avoidance circle as

$$\Delta x = tss \cdot cos(\theta) \cdot cos(\upsilon) \quad (13)$$

$$\Delta y = tss \cdot cos(\theta) \cdot sin(\upsilon) \quad (14)$$

$$\Delta z = tss \cdot \sin(\theta) \tag{15}$$

where *tss* is the threat sphere size, θ is the angle on the avoidance circle and υ is the angle between the ECI X axis and the debris position vector at the time of closest approach.

The following is a simple example of how to apply Equations (13-15).

1. $r_{deb} = [6652.34 \quad 0.0 \quad 0.0]$ is the ECI debris position vector at closest approach.
2. Since the debris position vector is along the ECI X axis the angle between the ECI X axis and the debris position vector (υ) is zero.
3. The avoidance circle location (θ) is 90° and the threat sphere size (*tss*) is 2 km.
4. Therefore,
$$\Delta x = tss \cdot \cos(\theta) \cdot \cos(\upsilon) = 2 \cdot \cos(90) \cdot \cos(0) = 0 \; km$$
$$\Delta y = tss \cdot \cos(\theta) \cdot \sin(\upsilon) = 2 \cdot \cos(90) \cdot \sin(0) = 0 \; km$$
$$\Delta z = tss \cdot \sin(\theta) = 2 \cdot \sin(90) = 2 \; km$$
5. The ECI position vector to the avoidance circle location can be calculated as
$$r_{ac} = [6652.34 + \Delta x \quad 0.0 + \Delta y \quad 0.0 + \Delta z] = [6652.34 \quad 0.0 \quad 2.0]$$

Figure (12) shows the ECI position vector to the debris and the ECI position vector to the 90° avoidance circle location.

The spacecraft position and velocity at the time of the maneuver can be found by propagating the initial spacecraft state vector to that time. The initial spacecraft velocity, along the proposed debris avoidance trajectory, at the time of the impulse is provided in the solution to Lambert's theorem. The difference in the two velocities at this time is the ΔV required to place the spacecraft on the debris avoidance trajectory and is expressed as

$$\Delta V = V_{at_{im}} - V_{sc_{im}} \tag{16}$$

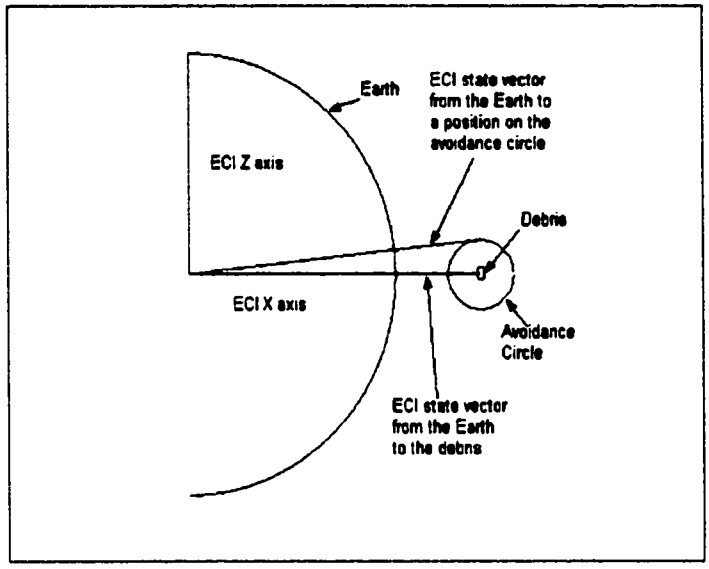


Figure (12) ECI Position Vector to an Avoidance Circle Location.

where V_{dt_m} is the velocity of the spacecraft on the avoidance trajectory at the maneuver time and $V_{sc_{t_m}}$ is the velocity of the spacecraft on its original orbit at the maneuver time. The state vectors from both orbits are used to determine the pre- and post-maneuver Keplerian orbital elements. The following is a list of the steps carried out in the single-impulse maneuver determination program.

1. Determine how long before the minimum separation distance the maneuver will take place.
2. Determine the position on the avoidance circle in ECI coordinates.
3. Solve Lambert's theorem from the initial position to the position on the avoidance circle.
4. Calculate the required ΔV to avoid the debris.
5. Calculate the orbital elements for both orbits for comparison.

The program output is a set of plots that specify the ΔV required to avoid the debris and the change in the orbital elements after the maneuver. All output is expressed as a function of the terminal location on the avoidance circle.

Figure (13) is the output plots for the scenario used in this study. These plots were generated for a maneuver that takes place thirty minutes (chosen arbitrarily) prior to minimum separation. The maximum change in the semimajor axis will be approximately the same as the threat sphere size. The threat sphere size for this study is arbitrarily set to 2 km. The minimum change in semimajor axis, zero, will occur with those maneuvers that have a maximum inclination change. The maximum inclination change was only 0.02°. However, this corresponds to a maximum ΔV of 2.7 meters per second (mps). The minimum ΔV of 1.46 mps occurs at those locations where there is no change in inclination. For this scenario those locations are 0° and 180° on the avoidance circle. The maximum change in eccentricity corresponds to the maximum change in the semimajor axis. However, like inclination, the change is very small. The maximum change in eccentricity was 0.0003 (eccentricity is unit less). The right ascension of the ascending node (RAAN) is undefined for orbits with zero inclination. The spacecraft orbit for this study has zero inclination therefore; the change in RAAN is simply the RAAN of the avoidance trajectory. The change in RAAN was 293° when targeting positions on the avoidance circle that were greater than 0° and less than 180°. The change in RAAN was 113° when targeting positions on the avoidance circle that were greater than 180° and less than 360°. When targeting 0° and 180° on the avoidance circle there is no inclination change, therefore, the inclination remains zero (equatorial orbit)

and the RAAN remains undefined. In Figure (13) the undefined points of the ΔRAAN curve are not displayed. The argument of perigee is undefined for circular orbits (eccentricity = 0). The spacecraft orbit and the avoidance trajectories are circular or near circular orbits. Therefore, the plot of the change in argument of perigee must be evaluated closely as it may contain invalid or misleading data. The change in argument of perigee was between 178° and 188° when the change in eccentricity was decreasing and between 0° and 10° when the change in eccentricity was increasing.

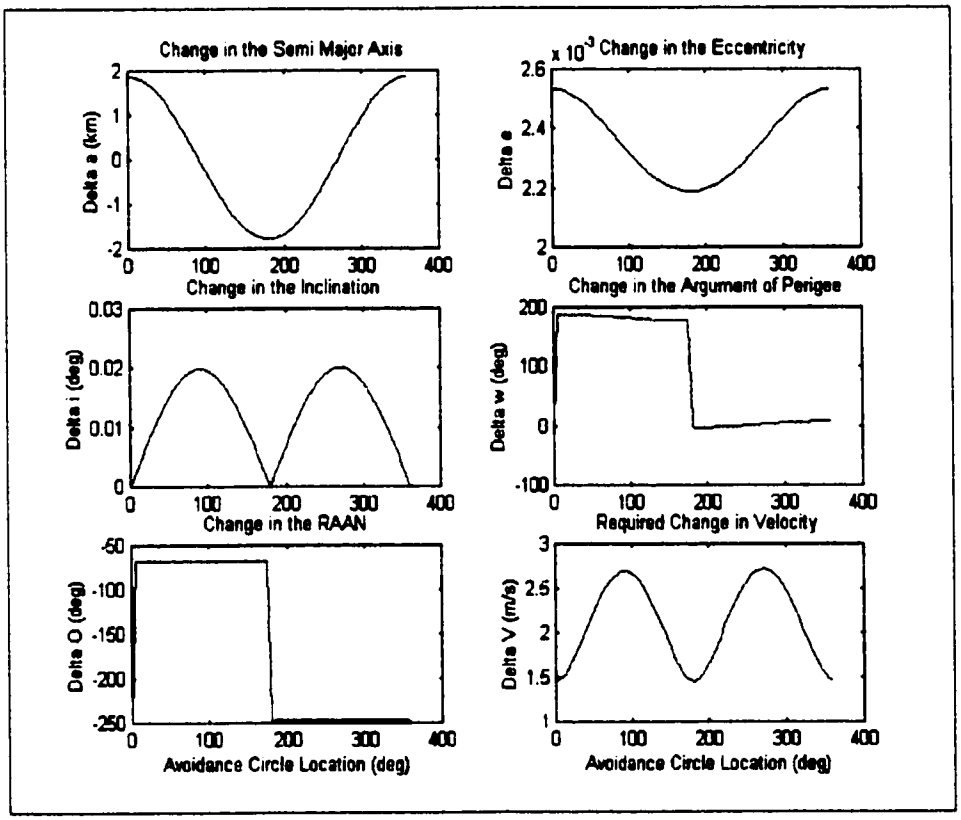


Figure (13) Single-Impulse Maneuver Output Plots.

As mentioned before, the output plots shown above are for maneuvers that take place thirty minutes prior to the minimum separation distance. If a different maneuver time were chosen, there would be a unique set of plots for that time. If the goal is to find

the absolute minimum ΔV, then the minimum ΔV for each set of plots must be determined. Figure (14) shows the minimum ΔV curve for the single-impulse maneuver case. A minimum ΔV of 1.46 m/sec was found when the maneuver was performed thirty minutes prior to the minimum separation distance.

If a single-impulse maneuver is desired, the graphical results would be evaluated along with the mission requirements and constraints to determine which, if any, single impulse maneuver is the best option.

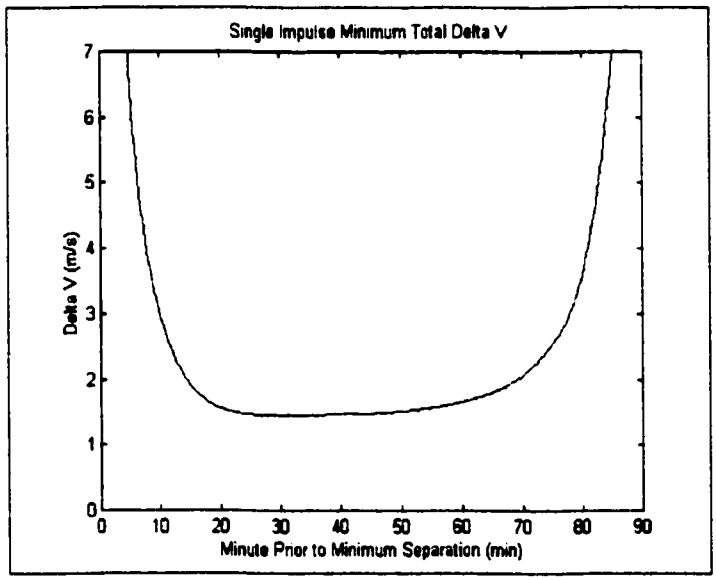


Figure (14) Single-Impulse Minimum ΔV Curve.

5.0 Two-Impulse Maneuver

Figure (15) shows the two-impulse maneuver portion of the debris avoidance flow chart. A program was written to calculate the two impulses required to avoid the debris and return the spacecraft to its original orbit (but not on-station). The input to the program is the number of hours, minutes, and seconds prior to the minimum separation distance that the maneuver will be performed. The first impulse is the same as that

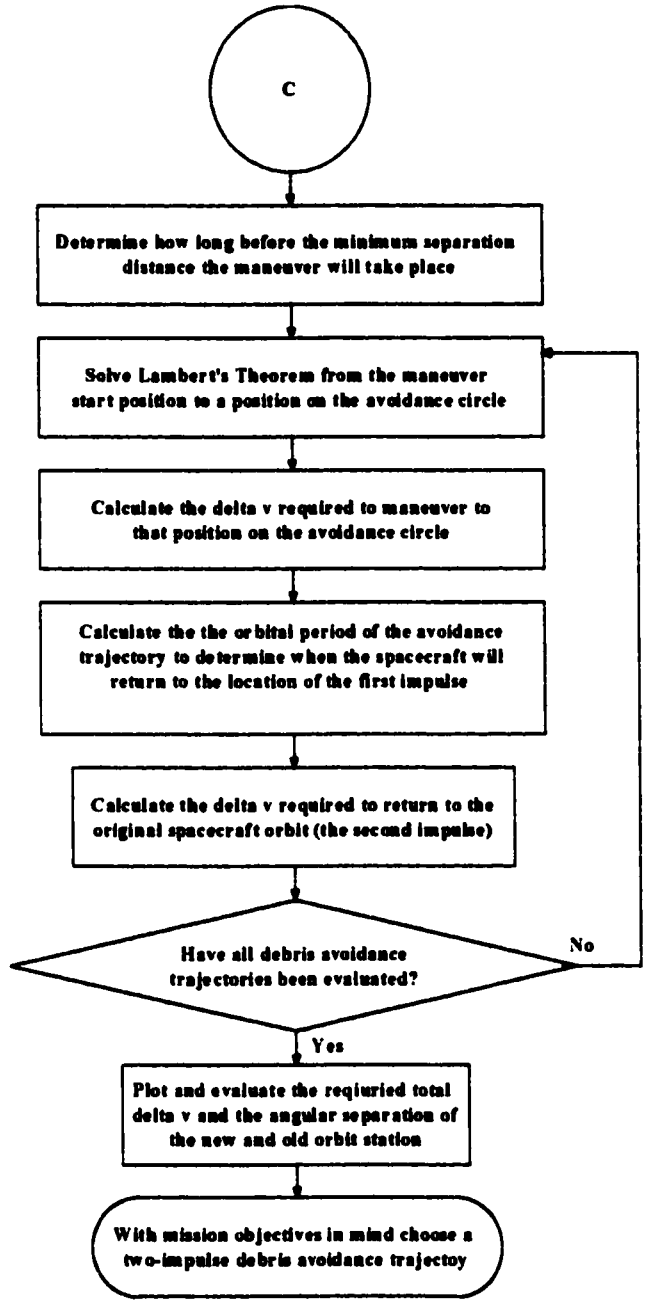


Figure (15) Debris Avoidance Procedure Flow Chart (Two-Impulse Maneuver Section).

described for the single-impulse maneuver case. The maneuver targets terminal positions around the avoidance circle at five-degree increments. Figure (10) shows the avoidance circle geometry. The ΔV for the first impulse is calculated by solving Lambert's theorem

(described in the Single-Impulse Maneuver section) from the maneuver location to the target point on the avoidance circle. As with the single-impulse case, the target points on the avoidance circle must be expressed in ECI coordinates. The ΔV required to place the spacecraft on the debris avoidance trajectory is expressed in Equations (16).

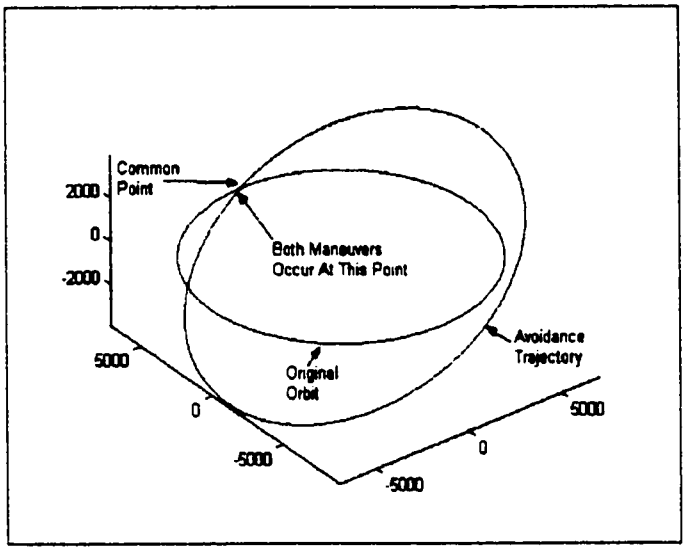


Figure (16) Common Position of the Spacecraft Orbit and Avoidance Trajectory.

The position where the maneuver takes place is a common point for the original spacecraft orbit and the debris avoidance trajectory (assuming two-body motion). Figure (16) illustrates this common position for the two orbits. After one orbital revolution on the avoidance trajectory the spacecraft will return to this common point. A second impulsive maneuver at this location then places the spacecraft on its original orbit. The ΔV required to return the spacecraft to its original orbit is expressed as

$$\Delta V = V_{oo_{m}} - V_{ad_{m}} \tag{17}$$

where $V_{oo_{m}}$ is the velocity on the original orbit at the common position at the time of the maneuver and $V_{ad_{t_{m}}}$ is the velocity of the spacecraft on the avoidance trajectory at the time

of the maneuver. The following is a list of the steps carried out in the program used to calculate the two-impulse avoidance maneuvers.

1. Determine how long before the minimum separation distance the maneuver will take place.
2. Determine the position on the avoidance circle in ECI coordinates.
3. Solve Lambert's theorem from the initial position to the position on the avoidance circle.
4. Calculate the required ΔV to avoid the debris.
5. Determine the orbital period of the avoidance trajectory.
6. Calculate the required ΔV to return to the original orbit after one period of travel on the avoidance trajectory.

The program output is a set of plots that specify the total ΔV required to avoid the debris and return to the original orbit. The angular separation between new station and the original station is also provided. The output is expressed as a function of the terminal location on the avoidance circle. Figure (17) displays plots for the scenario used in this study. These plots were generated for a maneuver in which the first impulse takes place thirty-six minutes prior to the minimum separation distance. A minimum total ΔV of 2.77 mps was found for those avoidance trajectories that required no inclination change. However, the minimum ΔV trajectory produced the largest angular separation between the original and new orbit station of 0.12 deg. A maximum total ΔV of 8.03 mps was found for those avoidance trajectories that required no semimajor axis change. Since there was no change in the semimajor axis for these trajectories, the periods of the

original orbit and the avoidance trajectory are the same (no angular separation between the original and new orbit station). Therefore, after the second impulse, the spacecraft will be back on its original orbit and its original station. It is also important to note that if there is no change in the semimajor axis there will be an earlier opportunity to return to the original orbit station. This opportunity occurs at one half the orbit period at the ascending or descending node depending on the inclination. (The ascending and descending nodes are the points where the spacecraft crosses the Earth's equatorial plane.)

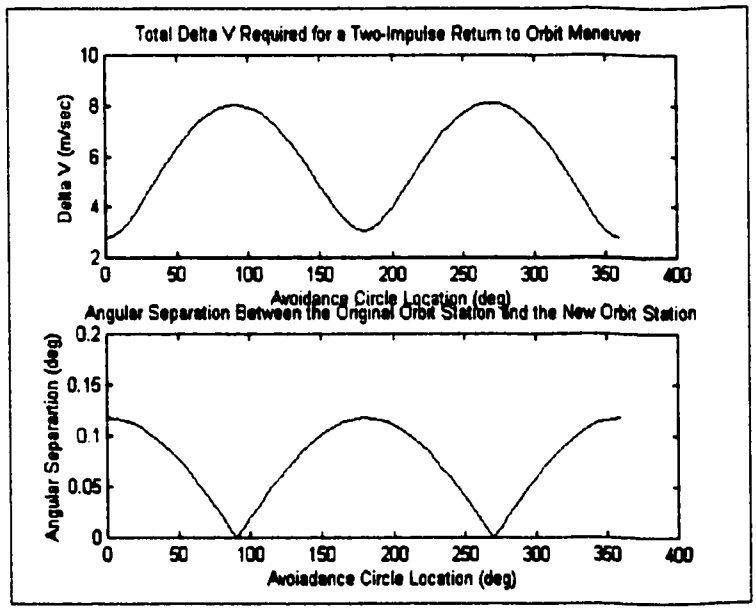


Figure (17) Two-Impulse Maneuver Output Plots.

The plots shown above are for maneuvers that take place thirty-six minutes prior to the minimum separation distance. If a different maneuver time was chosen there would obviously be a different set of plots for that time. There is a minimum total ΔV associated with each set of plots. Figure (18) shows the minimum total ΔV curve for the

two-impulse maneuver case. The minimum total ΔV is expressed as a function of the number of minutes prior to the minimum separation distance the first impulse takes place. A minimum total ΔV of 2.77 m/sec was found when the first maneuver was performed thirty-six minutes prior to the minimum separation distance. If it is necessary to remain on the original orbit but not the original station after an evasive maneuver, the two-impulse maneuver could be a simple and efficient method to do so.

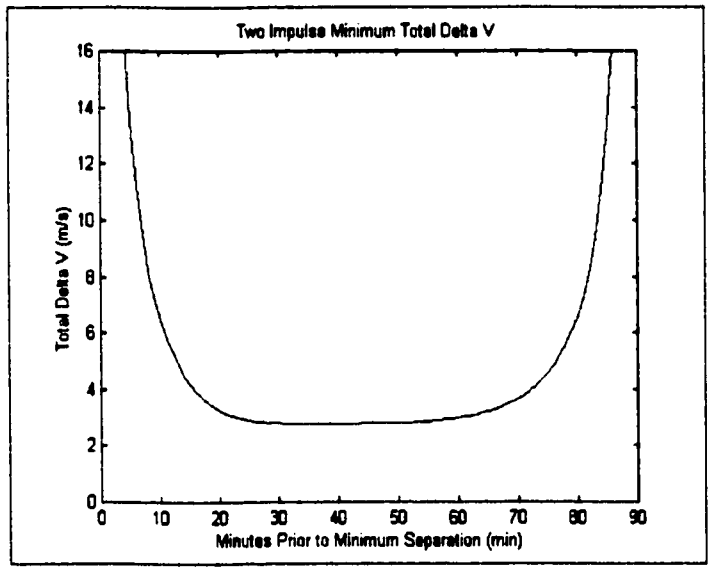


Figure (18) Two-Impulse Minimum ΔV.

6.0 Multiple-Impulse Maneuver

Figure (19) shows the multiple-impulse maneuver portion of the debris avoidance flow chart. A program was written to calculate the number impulses required to, *optimally* (minimum total ΔV), avoid the debris and return the spacecraft to its original orbit station. The input to the program is the number of hours, minutes, and seconds prior to the minimum separation distance that the maneuver will be performed and the targeted location on the avoidance circle. The multiple-impulse maneuver starts out as a

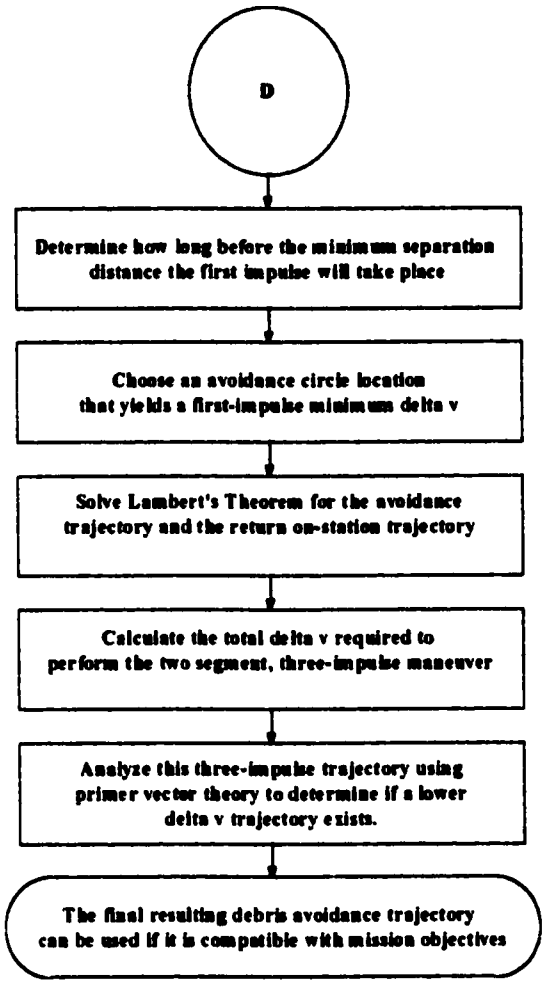


Figure (19) Debris Avoidance Procedure Flow Chart (Multiple-Impulse Section).

three-impulse maneuver. The first impulse puts the spacecraft on the debris avoidance trajectory. The second impulse puts the spacecraft on a trajectory back towards its original station and the third impulse will rendezvous with the original orbit station. This three-impulse trajectory is then analyzed using primer vector theory to determine if any changes to the trajectory will lower the total ΔV.

6.1 Initial Three-Impulse Trajectory

For the multiple-impulse return on-station scenario the objective is to find the

minimum total ΔV. The first impulse is the same as that described for the single-impulse case. For the single-impulse case the minimum ΔV was found for maneuvers that targeted 0° and 180° on the avoidance circle. For this case the first impulse targets the 0° location on the avoidance circle. The second impulse puts the spacecraft on the return on-station trajectory and is performed the instant the avoidance circle target is reached. Lambert's theorem must then be solved from the avoidance circle location to the station position. The time used in Lambert's theorem to determine the return on-station trajectory is changed iteratively until the trajectory that corresponds with the minimum ΔV is found. The third impulse is performed when the return on-station trajectory rendezvous with the original orbit station. Figure (20) shows the total ΔV as a function of the return on-station time. A minimum total ΔV of 5.274 m/s was found for a return on-station time of 74 minutes, and a total evasion time of 104 minutes. The three-impulse ΔVs are expressed as

$$\Delta V_1 = V_{ist} - V_{oo} \tag{18}$$

$$\Delta V_2 = V_{irt} - V_{fst} \tag{19}$$

$$\Delta V_3 = V_{os} - V_{frt} \tag{20}$$

where V_{oo} is the velocity on the original orbit, V_{ist} is the initial velocity on the avoidance trajectory, V_{fst} is the final velocity on the avoidance trajectory, V_{irt} is the initial velocity on the return trajectory, V_{frt} is the final velocity on the return trajectory and V_{os} is the velocity of the original station. $V_{ist}, V_{fst}, V_{irt}$, and V_{frt} are found from the solution to Lambert's theorem for each segment. This "minimum," three-impulse, debris avoidance

maneuver is now analyzed using primer vector theory to determine if any changes could be made to the maneuver that will yield a lower ΔV.

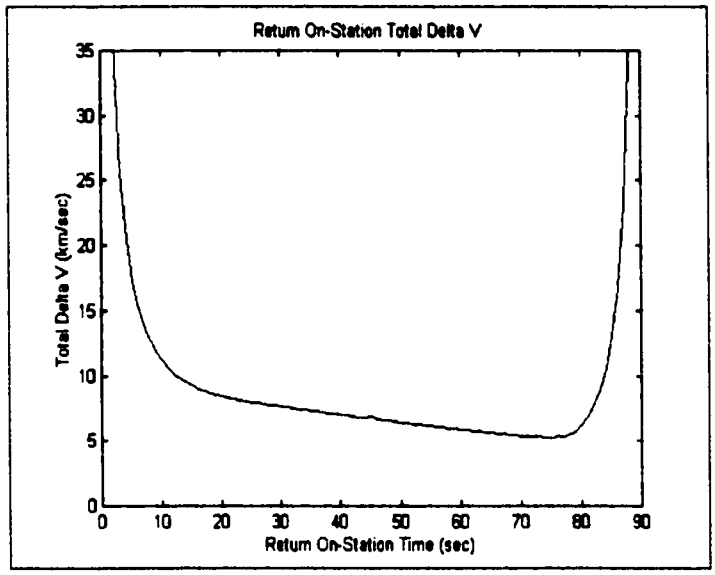


Figure (20) Three-Impulse Return On-Station Total ΔV.

6.2 Primer Vector Analysis

The necessary conditions for an optimized trajectory are described in terms of the primer vector. The primer vector is defined as the adjoint vector associated with the velocity. These necessary conditions were first introduced by Lawden[7].

1. The primer vector and its derivative must be continuous.
2. At an impulse time the primer vector is a unit vector in the optimal thrust direction.
3. The primer vector magnitude must not exceed unity on any trajectory segment.
4. The derivative of the primer vector must equal zero at all interior junction points.

Assume there is a trajectory Γ. Let $\boldsymbol{r}$ and $\boldsymbol{v}$ represent the position and velocity vectors along Γ and let $\boldsymbol{r}'$ and $\boldsymbol{v}'$ represent the position and velocity vectors along a

perturbed trajectory Γ'. Now we can define r' and v' as

$$r'(t) = r(t) + \delta r(t) \tag{21}$$
$$v'(t) = v(t) + \delta v(t) \tag{22}$$

where δr and δv are the change in the position and velocity vectors respectively.

If Γ and Γ' are close enough for linear analysis, then δr and δv are the solutions of the following system of equations.

$$\begin{bmatrix} \delta\dot{r} \\ \delta\dot{v} \end{bmatrix} = \begin{bmatrix} 0 & I \\ G & 0 \end{bmatrix} \begin{bmatrix} \delta r \\ \delta v \end{bmatrix} \tag{23}$$

where I is the 3 x 3 identity matrix and G is the gravity gradient matrix which includes all gravitational effects under consideration. Equation (23) is in first-order form and can be written in second-order form as

$$\delta\ddot{r} = G\delta r \tag{24}$$

To find the adjoint vector, the Hamiltonian (H) of the system must be found. The equations of motion for the spacecraft can be written as

$$\frac{dr}{dt} = \dot{r} = v \tag{25}$$
$$\frac{dv}{dt} = \dot{v} = g \tag{26}$$

where g is the gravitational acceleration vector.

The Hamiltonian for this system is the summation of each adjoint vector multiplied by the right side of its corresponding state equation. The Hamiltonian is expressed as

$$H = \eta' v + \lambda' g \tag{27}$$

where η is the adjoint vector associated with the position vector and λ is the adjoint

vector associated with the velocity vector. Now the adjoint system of equations can be found in terms of the Hamiltonian as

$$\dot{\eta} = -\frac{\partial H}{\partial r} = \frac{\partial g}{\partial r} = -G\lambda \tag{28}$$

$$\dot{\lambda} = -\frac{\partial H}{\partial v} = -\eta \tag{29}$$

This first order system can also be written as

$$\begin{bmatrix} \dot{\eta} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} 0 & -G \\ -I & 0 \end{bmatrix} \begin{bmatrix} \eta \\ \lambda \end{bmatrix} \tag{30}$$

or in second order form as

$$\ddot{\lambda} = G\lambda \tag{31}$$

The system from Equations (24) and (31) are identical. The equations that are numerically integrated to obtain the state transition matrix (STM) for both systems are also identical. Therefore, the STM for both systems will be the same. The STM is defined as

$$STM = \Phi(t,t_0) = \begin{bmatrix} \frac{\partial x}{\partial x_0} & \frac{\partial x}{\partial y_0} & \frac{\partial x}{\partial z_0} & \frac{\partial x}{\partial \dot{x}_0} & \frac{\partial x}{\partial \dot{y}_0} & \frac{\partial x}{\partial \dot{z}_0} \\ \frac{\partial y}{\partial x_0} & \frac{\partial y}{\partial y_0} & \frac{\partial y}{\partial z_0} & \frac{\partial y}{\partial \dot{x}_0} & \frac{\partial y}{\partial \dot{y}_0} & \frac{\partial y}{\partial \dot{z}_0} \\ \frac{\partial z}{\partial x_0} & \frac{\partial z}{\partial y_0} & \frac{\partial z}{\partial z_0} & \frac{\partial z}{\partial \dot{x}_0} & \frac{\partial z}{\partial \dot{y}_0} & \frac{\partial z}{\partial \dot{z}_0} \\ \frac{\partial \dot{x}}{\partial x_0} & \frac{\partial \dot{x}}{\partial y_0} & \frac{\partial \dot{x}}{\partial z_0} & \frac{\partial \dot{x}}{\partial \dot{x}_0} & \frac{\partial \dot{x}}{\partial \dot{y}_0} & \frac{\partial \dot{x}}{\partial \dot{z}_0} \\ \frac{\partial \dot{y}}{\partial x_0} & \frac{\partial \dot{y}}{\partial y_0} & \frac{\partial \dot{y}}{\partial z_0} & \frac{\partial \dot{y}}{\partial \dot{x}_0} & \frac{\partial \dot{y}}{\partial \dot{y}_0} & \frac{\partial \dot{y}}{\partial \dot{z}_0} \\ \frac{\partial \dot{y}}{\partial x_0} & \frac{\partial \dot{y}}{\partial y_0} & \frac{\partial \dot{y}}{\partial z_0} & \frac{\partial \dot{y}}{\partial \dot{x}_0} & \frac{\partial \dot{y}}{\partial \dot{y}_0} & \frac{\partial \dot{y}}{\partial \dot{z}_0} \end{bmatrix} \tag{32}$$

and can be partitioned into four 3 x 3 matrices as

$$\Phi = \begin{bmatrix} \Phi_{11}(t,\tau) & \Phi_{12}(t,\tau) \\ \Phi_{21}(t,\tau) & \Phi_{22}(t,\tau) \end{bmatrix} \tag{33}$$

λ from hereafter will be referred to as the primer vector $\boldsymbol{p}$. The primer vector and its derivative transform the same way the position and velocity, so the solution of Equation (31) is

$$\begin{bmatrix} \boldsymbol{p}(t) \\ \dot{\boldsymbol{p}}(t) \end{bmatrix} = \begin{bmatrix} \Phi_{11}(t,t_0) & \Phi_{12}(t,t_0) \\ \Phi_{21}(t,t_0) & \Phi_{22}(t,t_0) \end{bmatrix} \begin{bmatrix} \boldsymbol{p}(t_0) \\ \dot{\boldsymbol{p}}(t_0) \end{bmatrix} \tag{34}$$

Recalling from the necessary conditions, at the time of an impulse, the primer vector is a unit vector in the trust direction. Therefore, on a trajectory from t_0 to t_f, the primer vector satisfies Equation (34) with boundary conditions

$$\boldsymbol{p}(t_0) = \frac{\Delta \boldsymbol{V}_0}{|\Delta \boldsymbol{V}_0|} \tag{35}$$

$$\boldsymbol{p}(t_f) = \frac{\Delta \boldsymbol{V}_f}{|\Delta \boldsymbol{V}_f|} \tag{36}$$

where $\Delta \boldsymbol{V}_0$ and $\Delta \boldsymbol{V}_f$ are the impulsive maneuvers at t_0 and t_f respectively.

To use Equation (34) to find the primer magnitude at any given time, the derivative of the primer vector $\dot{\boldsymbol{p}}$ at t_0 is also required and is expressed as

$$\dot{\boldsymbol{p}}(t_0) = \Phi_{12}^{-1}(t_f,t_0)\left[\boldsymbol{p}(t_f) - \Phi_{11}(t_f,t_0)\boldsymbol{p}(t_0)\right] \tag{37}$$

The primer vector magnitude history provides information on how to improve a two-impulse trajectory. An improvement is a change to the trajectory that yields a lower ΔV. Anytime a change is made to the trajectory the primer vector magnitude history must be evaluated to determine, once again, if any improvements can be made.

Figure (21) shows a fabricated sample of primer vector magnitude histories.

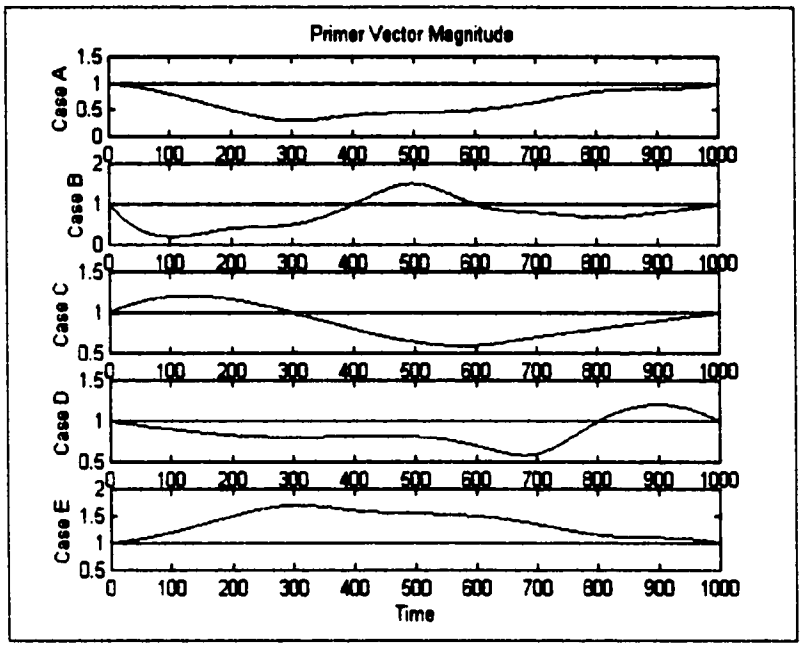


Figure (21) Primer Vector Magnitude History Samples.

The primer vector history in Case A represents an optimal trajectory. The primer magnitude remains less than or equal to unity along the entire trajectory.

In Case B the primer magnitude exceeds unity somewhere between the initial and final impulse. A trajectory with a lower ΔV exists if an additional impulse is added at the time t_m (the subscript m refers to midcourse) where the primer magnitude reaches a maximum and in the direction of primer vector. If the state transition matrix is evaluated from t_0 to t_m and from t_f to t_m, then the following relationships can be made using standard linearization theories.

$$\delta v_{m-} = \Phi_{22}(t_m, t_0)\delta v_0 \tag{38}$$

$$\delta v_{m+} = \Phi_{22}(t_m, t_f)\delta v_f \tag{39}$$

$$\delta v_0 = \Phi_{12}^{-1}(t_m, t_0)\delta r_m \tag{40}$$

$$\delta v_f = \Phi_{12}^{-1}(t_m, t_f)\delta r_m \tag{41}$$

where – and + refer to times just before and after the midcourse time respectively. Using Equations (38-41), the velocity increment can be written as

$$\delta v_{m+} - \delta v_{m-} = A\delta r_m \tag{42}$$

where

$$A = \Phi_{22}(t_m, t_f)\Phi_{12}^{-1}(t_m, t_f) - \Phi_{22}(t_m, t_0)\,\Phi_{12}^{-1}(t_m, t_0) \tag{43}$$

The change in the position vector, δr_m, at time t_m, which will define the perturbed trajectory Γ', can be calculated as.

$$\delta r_m = cA^{-1}\frac{p_m}{|p_m|} \tag{44}$$

where c is the magnitude of the midcourse impulse and is quantified by Jezewski and Rozendaal[15] as

$$c = \frac{\dfrac{\beta \cdot \Delta V_f}{|\Delta V_f|} - \dfrac{\alpha \cdot \Delta V_0}{|\Delta V_0|} - 1}{\dfrac{\left[\alpha \cdot \alpha - \dfrac{(\alpha \cdot \Delta V_0)^2}{|\Delta V_0|^2}\right]}{|\Delta V_0|} + \dfrac{\left[\beta \cdot \beta - \dfrac{(\beta \cdot \Delta V_f)^2}{|\Delta V_f|^2}\right]}{|\Delta V_f|}} \tag{45}$$

where

$$\alpha = \Phi_{12}^{-1}(t_m, t_0)A^{-1}\frac{p_m}{|p_m|} \tag{46}$$

$$\beta = \Phi_{12}^{-1}(t_m, t_f)A^{-1}\frac{p_m}{|p_m|} \tag{47}$$

Now Lambert's theorem is solved from r_0 to $r_m^{\Gamma'}$ and from $r_m^{\Gamma'}$ to r_f. The total ΔV of

this perturbed trajectory Γ' should be less than the reference trajectory Γ. However, in general this trajectory will not be optimized. The primer magnitude history at the time of the mid-course impulse may form a cusp. A cusp at the impulse time indicates that the derivative of the primer vector is not continuous and that a three-impulse trajectory with a lower ΔV exists. In this scenario an optimization algorithm must be used to iterate on $\boldsymbol{r}_m^{\Gamma'}$ and t_m until the minimum ΔV three-impulse trajectory is found. The optimization algorithm must minimize the cost function

$$J = \Delta V_0 + \Delta V_m + \Delta V_f \tag{48}$$

As done in References (16), the gradients of the cost function can be found in terms of the primer vector. An optimization technique that uses the gradient components can be used to minimize the cost function J. The gradient of the cost function with respect to the midcourse position and time are

$$\frac{\partial J}{\partial \boldsymbol{r}_m} = (\boldsymbol{p}_m^+ - \boldsymbol{p}_m^-) \tag{49}$$

$$\frac{\partial J}{\partial t_m} = -(\boldsymbol{p}_m^{+\mathrm{T}} \boldsymbol{v}_m^+ - \boldsymbol{p}_m^{-\mathrm{T}} \boldsymbol{v}_m^-) \tag{50}$$

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, recommended by Reference (16), is an unconstrained, nonlinear, gradient method. The inputs to the algorithm are $\boldsymbol{r}_m^{\Gamma'}$, t_m, and the gradients from Equations (49-50). The output is a new value for $\boldsymbol{r}_m^{\Gamma'}$ and t_m that will yield a lower ΔV three-impulse trajectory. Iterations continue until the gradients are equal to zero or the decrease in the cost function is negligible. The BFGS algorithm for programming purposes can be found in Reference (10). The BFGS

method was used because of its coding simplicity.

The primer magnitude in Case C exceeds unity just after the first impulse. A lower ΔV trajectory can be found if the spacecraft is allowed to "coast" on its original orbit for some duration before the first impulse is performed. The coast time that yields the minimum ΔV can be found by changing the coast time iteratively until the minimum is found.

In Case D the primer magnitude exceeds unity just before the final impulse. If the final impulse is performed some time prior to its original intended execution time an improvement to the trajectory will be made. As with Case C this time is changed iteratively until the minimum ΔV is found.

In Case E the primer magnitude exceeds unity along the entire trajectory. With this scenario any of the changes mentioned above will lower the total ΔV.

As mentioned before the initial debris avoidance trajectory described in this section is a two-segment, three-impulse maneuver. The first segment is the avoidance trajectory and the second segment is the return on-station trajectory. Primer vector theory is not applied to the first segment. Any changes to the avoidance trajectory could result in a threat sphere violation. The primer vector magnitude history for the initial three-impulse maneuver is displayed in Figure (22). The total ΔV was 5.274 m/s.

The primer magnitude starts to exceed unity 31 minutes into the return on-station trajectory. This case is similar to Case B where an additional impulse at the time where the primer magnitude reaches a maximum will lower the total ΔV. The primer magnitude reaches a maximum of 1.8561 at 51 minutes into the return on-station trajectory.

A third impulse was added and the corresponding total ΔV was 5.060 m/s. This is an improvement of 0.214 m/s over the initial three-impulse maneuver. Figure (23) shows the primer vector magnitude history for the new three-segment, four-impulse debris

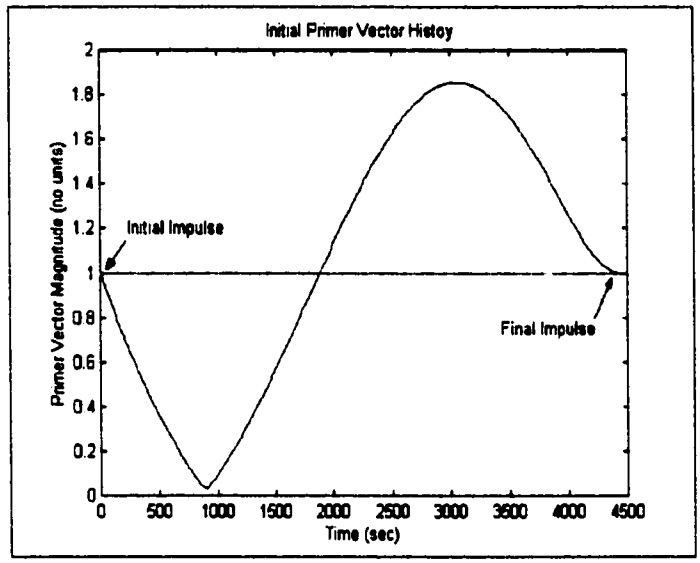


Figure (22) Initial Three-Impulse Primer Vector Magnitude History.

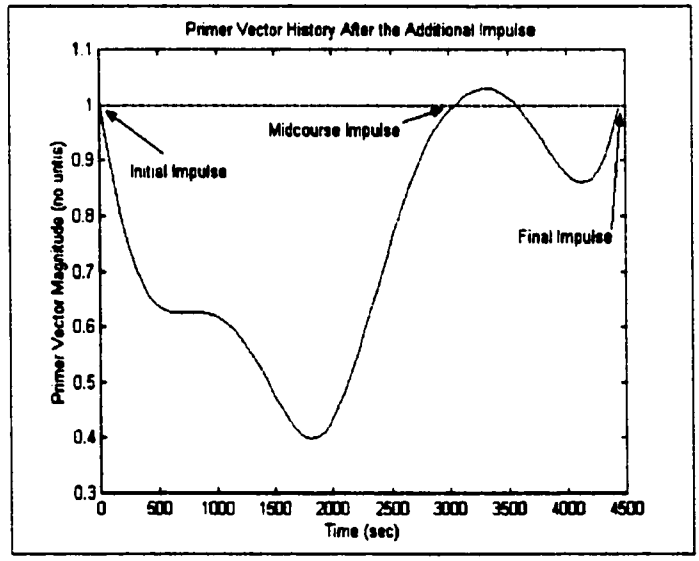


Figure (23) Four-Impulse Trajectory Primer Magnitude History.

avoidance maneuver. Once again the primer magnitude exceeds unity. However, this

time the primer magnitude exceeds unity just after the first impulse of the third segment. This is similar to Case C where if the spacecraft is allowed to coast through the intended impulse location, and perform the impulsive burn some time after; a further reduction in the total ΔV is possible. A total minimum ΔV of 5.056 m/s was found when the spacecraft was allowed to coast for 70 seconds before the first impulse of the third segment was performed. Figure (24) is the primer magnitude history for the final optimized debris avoidance trajectory. The necessary conditions for an optimal maneuver have now been satisfied.

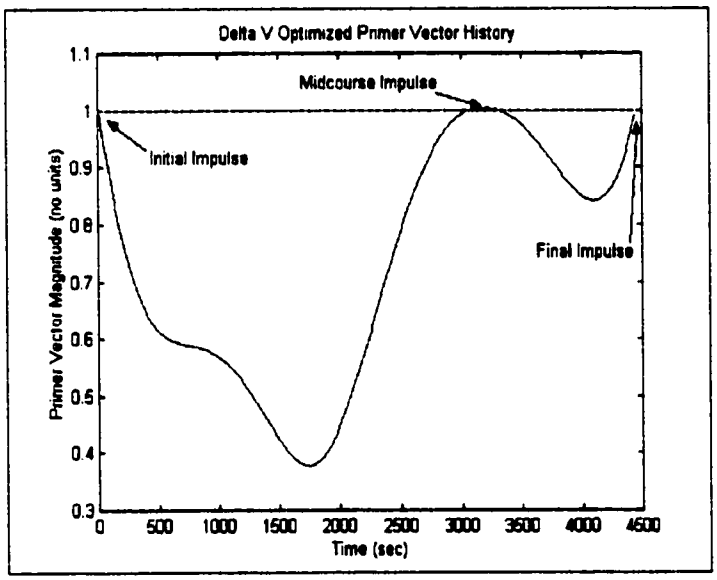


Figure (24) Optimized Four-Impulse Trajectory Primer Magnitude History.

For a spacecraft, in a station specific orbit, that must perform a debris avoidance maneuver, the optimized return on-station maneuver will allow the spacecraft to maintain its original orbit configuration. However, the cost of this type of maneuver is the most expensive of the three cases evaluated in this study.

7.0 Results Summary

For the single impulse debris avoidance maneuver the minimum ΔV was 1.46 m/s when the maneuver was performed thirty minutes prior to minimum separation. The single impulse maneuver can be used if the spacecraft's mission objectives can be met in an orbit that differs slightly from the original. The orbit following the avoidance maneuver differs by 1.78 km in semimajor axis and 0.0022 in eccentricity. If a change in semimajor axis is not acceptable a 0.02° inclination change maneuver can be performed at cost of 2.7 m/s.

If the mission objectives can only be met using the current orbit the two-impulse return to orbit maneuver should be used. The minimum total ΔV for this maneuver was 2.77 m/s when the first impulse was performed 36 minutes prior to minimum separation. After the two-impulse maneuver is performed an angular separation of 0.12° results between the original and new orbit stations. However, there is a two-impulse maneuver that will return the spacecraft to its original station. The total ΔV for a two-impulse return on-station maneuver is 8.03 m/s - a considerable increase in cost. If a return on-station maneuver is required a multiple impulse maneuver should be considered.

The minimum total ΔV for the initial three-impulse return on-station maneuver was 5.274 m/s. After primer vector analysis an additional impulse was added and the total ΔV was reduced to 5.056 m/s. This is a significant improvement over the two-impulse return on-station maneuver. The total evasion time for this maneuver was 104 minutes. If this duration is not acceptable a three-impulse maneuver with a lower return on-station time can be used. However, as the return on-station time decreases the total

ΔV increases. Figure (20) shows the relationship between return on-station time and total ΔV. The thirty minute duration of the avoidance trajectory must be added to the return on-station time to obtain the total evasion time. The following table is a summary of the minimum ΔV debris avoidance maneuvers.

	Single-Impulse Maneuver	Two-Impulse Maneuver	Multiple-Impulse Maneuver
Variable Orbit Mission	**1.46 m/s**	Not Evaluated	Not Evaluated
Orbit Specific Mission	N/A	**2.77 m/s**	Not Evaluated
Station Specific Mission	N/A	**8.03 m/s**	**5.06 m/s**

Table (1) Summary of the Minimum ΔV Debris Avoidance Maneuvers

It is important to keep in mind that these results are for the conditions specified in this study. Obviously the results for debris avoidance will differ in orbits with different orbital elements.

8.0 Recommendations For Future Work

As mentioned above, the results generated in this thesis are for a specific debris avoidance case. Therefore, there are many opportunities for further studies.

A potential study would be to determine if the procedure outlined in this thesis can be extended to different orbital geometries. An entirely new procedure may have to be developed, or the current procedure may only need modifications.

Another interesting study would be a comparison of actual debris avoidance maneuvers to the maneuvers generated in this study. This will help establish whether or not the approach taken in this thesis produces practical debris avoidance strategies.

It may be beneficial to study the orbits of some of the larger tracked pieces of debris to determine if the objects will reenter the Earth's atmosphere and be destroyed or if they will continue to be a threat in space.

Any future debris avoidance studies will be important as the increased use of outer space will continue to produce potentially hazardous objects in space. In this author's point of view, actions must be taken to minimize the creation of space debris and establish a safer near Earth environment for satellites and human space flight.

9.0 Concluding Remarks

Using orbital mechanics and optimization techniques a procedure was developed that generates and analyzes many debris avoidance maneuver options. The procedure successfully calculates minimum fuel avoidance maneuvers as well as other maneuver options.

Regardless of whether a debris avoidance maneuver will be performed or not, the orbit of the debris in question must be evaluated thoroughly to determine if any future threat exists. The debris could be in an orbit that will continue to threaten the spacecraft's mission. Imminent danger requires immediate action, however, long term trending is equally important.

Planning for the minimum ΔV maneuver could be dangerous. In the single-impulse case a minimum total ΔV of 1.46 m/s when the maneuver was performed thirty minutes prior to the minimum separation event. If there was a ten minute delay in executing the maneuver, the ΔV required to avoid the debris increases to 1.56 m/s. If the maneuver was planed to execute forty minutes prior to the minimum separation the total

ΔV is 1.47 m/s and will continue to decrease (only slightly) for the next ten minutes. It is always a good idea, whenever possible, to allocate some time for contingency.

During a spacecraft's lifetime an orbit debris avoidance maneuver may be required to preserve mission life. Mission objectives and spacecraft capabilities will be the driving force in determining which avoidance maneuver to perform. The debris avoidance procedure outlined in this study may assist in making a practical decision.

References

[1]United States Space Command., *Satellite Box Score*, World Wide Web, http://www.peterson.af.mil/usspace/boxscore.htm, 2000.

[2]NASA Johnson Space Center., *Frequently Asked Questions About Orbital Debris*, World Wide Web, http://sn-callisto.jsc.nasa.gov/faq/faq.html#3, 2000.

[3]NASA Johnson Space Center., *Orbital Debris Graphics*, World Wide Web, http://sn-callisto.jsc.nasa.gov/beehives.html, 1999.

[4]Center for Orbital and Reentry Debris Studies., *What is Orbital Debris*, World Wide Web, http://www.aero.com/cords/orbdebris.html, 1995-2000.

[5]Chobotov, V. A., *Orbital Mechanics*, American Institute of Aeronautics and Astronautics, Washington, D.C., 1991.

[6]Prussing, J. E., and Clifton R. S., "Optimal Multiple-Impulse Satellite Evasive Maneuvers," *Journal of Guidance, Control, and Dynamics*, Vol. 17, No. 3, 1994, pp. 599-606.

[7]Lawden, D. F., *Optimal Trajectories for Space Navigation*, Butterworths, London, 1963.

[8]Lion, P. M., and Handelsman, M., "Primer Vector on Fixed-Time Impulsive Trajectories," *AIAA Journal*, Vol. 6, No. 1, 1968, pp. 127-132.

[9]Bryson, A. E., and Ho, Y.-C., *Applied Optimal Control*, Hemisphere Publishing, 1975.

[10]H.J. Greenberg., *Mathematical Programming Glossary*, World Wide Web, http://www.cudenver.edu/~hgreenbe/glossary/glossary.html, 1996-2000.

[11]D'Amario, L. A., "Minimum Impulse Three-Body Trajectories," Ph.D. Dissertation, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 1973.

[12]Commitee on Space Shuttle Meteoroid/Debris Risk Management, and Aeronautics and Space Engineering Board, *Protecting the Space Shuttle from Meteoroids and Orbital Debris*, National Academy Press, Washington, D.C., 1997.

[13]Bate, R. R., Mueller, D. D., and White, J. E., *Fundamentals of Astrodynamics*, Dover Publications, New York, 1971.

[14]Shepperd, S. W., "Universal Keplerian State Transition Matrix," *Celestial Mechanics*, Vol. 35, 1985, pp. 129-144.

[15]Jezewski, D. J., Rozendaal, H. L., "An Efficient Method for Calculating Optimal Free-Space *N*-Impulse Trajectories," *AIAA Journal*, Vol. 6, No. 11, 1968, pp. 2160-2165.

[16]Hiday, L. A., Howell, K. C. "Transfers Between Libration-Point Orbits in the Elliptic Restricted Problem," School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN.

[17]M. Compere., *Octave* <-> *Matlab Compatibility Database*, World Wide Web, http://users.powernet.co.uk/kienzle/octave/matcompat, 2000-2001.

Appendix A

The following is a list of the script files used to produce the results (using MATLAB®) shown in this thesis. The files in bold type were created by the author of this thesis. The italicized files were provided in the Orbital Mechanics MATLAB® Tool Box from Science Software. The underlined file was created based on ideas from Science Software. rk4fixed.m was created by Marc Compere and obtained from Reference (17).

catdata.m	Loads initial conditions
constants.m	Load general constants
cow.m	Contains equations of motion for numerical integration
eci2orb1.m	Converts Cartesian elements to Keplerian elements
gdate2.m	Convert Julian date to calendar date
j2stm.m	Contains equations for State Transition Matrix numerical integration
julian.m	Converts calendar date to Julian Date
search.m	Calculate close approach information
moon.m	Calculates Moon position for a given Julian Date
piter.m	Solves Lambert's theorem
plamb.m	Updates the solution of piter.m for perturbed orbits
sun.m	Calculates Sun position for a given Julian Date
pvhist.m	Generates primer vector magnitude history
pvinit.m	Calculates initial primer vector derivative

rk4fixed.m	Fourth order Runge-Kutta fixed step numerical integrator
spmans.m	Calculates single-impulse debris avoidance maneuvers
strdate.m	Converts numerical date to a character string
trespmans.m	Calculates multiple-impulse debris avoidance maneuvers
twopmans.m	Calculates two-impulse debris avoidance maneuvers

The following are the actual MATLAB® script files described in the list above.

```
catdata.m
% Initial Conditions File

%Enter the spacecraft's orbital parameters

% Date and Time
year1 = 2001;
month1 = 1;
day1 = 1;
hour1 = 12;
min1 = 0;
sec1 = 0;

time1 = [day1 month1 year1 hour1 min1 sec1];

% Position and Velocity
xcom1 = 6572.1355376203226;
ycom1 = 1026.81876109079;
zcom1 = 0.00076040479;
xdcom1 = -1.19682462988;
ydcom1 = 7.64800109829;
zdcom1 = 0.00000047722;

orb1 = [xcom1 ycom1 zcom1 xdcom1 ydcom1 zdcom1];


% Enter the debris' orbital parameters

% Date and Time
year2 = 2001;
month2 = 1;
day2 = 1;
hour2 = 12;
min2 = 22;
sec2 = 30;

time2 = [day2 month2 year2 hour2 min2 sec2];

% Position and Velocity
xcom2 = -1156.67993510388;
ycom2 = 0.00036531282;
zcom2 = 6548.22225323210;
xdcom2 = -7.61728033222;
ydcom2 = -0.00000073574;
zdcom2 = -1.34337020408;

orb2 = [xcom2 ycom2 zcom2 xdcom2 ydcom2 zdcom2];

% Enter close approach search start time

day =   1;
month = 1;
year =  2001;
hour =  12;
min =   20;
sec =   0;

sdate = julian(month,day,year) + hour/24 + min/1440 + sec/86400;

.....

% Constants

global dtr rtd atr
dtr = pi / 180;
rtd = 180 / pi;
atr = dtr / 3600;

global isun imoon idrag isrp
isun  = 1;
imoon = 1;
idrag = 0;
isrp  = 0;

global flat req mu lgrav mgrav j2 c2 c3
flat = 1 / 298.257;
req = 6378.14;
mu = 398600.5;
lgrav = 2;
mgrav = 0;
[ccoef, sccef] = readegm('egm96.dat');
j2 = -ccoef(3, 1);
c2 = mu*j2*req*req;
c3 = mu*j2*req*req*req;

global smu mmu aunit
smu = 132712438000;
mmu = 4902.793;
aunit = 149597870;
```

```
global rkcoef tetol neq stpsz options
rkcoef = 1;
neq = 6;
tetol = 1.0e-8;
stpsz = 1;
options = odeset('RelTol',1e-9,'AbsTol',1e-9);
```

```
% cow.m
% Cowell equations of motion
function xdot =  cow(t,x);

% Initialize global variables
global jdate isun imoon idrag isrp
global smu mmu j2 req mu bigc

% Initialize other variables
xdot = zeros(6,1);
asun  = zeros(3,1);
amoon = zeros(3,1);
agrav = zeros(3,1);

% Upate Julian date

idate = jdate + t/86400;

% Calculate the J2 perturbation for the current time step
r2 = x(1) * x(1) + x(2) * x(2) + x(3) * x(3);
r1 = sqrt(r2);
r3 = r2 * r1;
r5 = r2 * r3;
d1 = -1.5 * j2 * req * req * mu / r5;
d2 = 1 - 5 * x(3) * x(3) / r2;

agrav(1) = agrav(1) + x(1) * d1 * d2;
agrav(2) = agrav(2) + x(2) * d1 * d2;
agrav(3) = agrav(3) + x(3) * d1 * (d2 + 2);


% Calculate Solar gravity perturbation for the current time step
[rasc, decl, rsun] = sun(idate);
rsunm = sqrt(dot(rsun, rsun));

for i = 1:1:3
    rs2sc(i) = x(i) - rsun(i);
end

rs2scm = sqrt(dot(rs2sc, rs2sc));
rrs2sc = smu / rs2scm ^ 3;
rrsun = smu / rsunm ^ 3;

for i = 1:1:3
    asun(i) = -rs2sc(i) * rrs2sc - rsun(i) * rrsun;
end

% Calculate the Lunar perturbation for the current time step
[rass, decl, rmoon] = moon(idate);
rmoonm = sqrt(dot(rmoon, rmoon));

for i = 1:1:3
    rm2sc(i) = x(i) - rmoon(i);
end

rm2scm = sqrt(dot(rm2sc, rm2sc));
rrm2sc = mmu / rm2scm^3;
rrmoon = mmu / rmoonm^3;

for i = 1:1:3
    amoon(i) = -rm2sc(i) * rrm2sc - rmoon(i) * rrmoon;
end


for i = 1:1:3
   perturb(i) =  asun(i) + amoon(i) + agrav(i);
end

% Return the state vector for the current time step
xdot(1) = x(4);
xdot(2) = x(5);
xdot(3) = x(6);
xdot(4) = perturb(1) - (mu*x(1)/(r1^3));
xdot(5) = perturb(2) - (mu*x(2)/(r1^3));
xdot(6) = perturb(3) - (mu*x(3)/(r1^3));
```

```
function oev = eci2orbl (r, v)
```

```
% convert eci state vector to six
% classical orbital elements via
% equinoctial elements

% input

%  r  = eci position vector (kilometers)
%  v  = eci velocity vector (kilometers/second)

% output

%  oev(1)  = semimajor axis (kilometers)
%  oev(2)  = orbital eccentricity (non-dimensional)
%            (0 <= eccentricity < 1)
%  oev(3)  = orbital inclination (radians)
%            (0 <= inclination <= pi)
%  oev(4)  = argument of perigee (radians)
%            (0 <= argument of perigee <= 2 pi)
%  oev(5)  = right ascension of ascending node (radians)
%            (0 <= raan <= 2 pi)
%  oev(6)  = true anomaly (radians)
%            (0 <= true anomaly <= 2 pi)

% Orbital Mechanics MATLAB Toolbox

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global mu

% position and velocity magnitude

rmag = sqrt(dot(r, r));

vmag = sqrt(dot(v, v));

% position an  velocity unit vectors

rhat = uvector(r);

vhat = uvector(v);

% angular momentum vectors

hv = cross(r, v);

hhat = uvector(hv);

% eccentricity vector

vtmp = v / mu;

ecc = cross(vtmp, hv);

ecc = ecc - rhat;

% semimajor axis

sma = 1 / (2 / rmag - vmag * vmag / mu);

p = hhat(1) / (1 + hhat(3));

q = -hhat(2) / (1 + hhat(3));

const1 = 1 / (1 + p * p + q * q);

fhat(1) = const1 * (1 - p * p + q * q);
fhat(2) = const1 * 2 * p * q;
fhat(3) = -const1 * 2 * p;

ghat(1) = const1 * 2 * p * q;
ghat(2) = const1 * (1 + p * p - q * q);
ghat(3) = const1 * 2 * q;

h = dot(ecc, ghat);

xk = dot(ecc, fhat);

x1 = dot(r, fhat);

y1 = dot(r, ghat);

% orbital eccentricity

eccm = sqrt(h * h + xk * xk);

% orbital inclination

inc = 2 * atan(sqrt(p * p + q * q));
```

```
% true longitude

xlambdat = atan3(y1, x1);

% check for equatorial orbit

if (inc > 0.00000001)
   raan = atan3(p, q);
else
   raan = 0;
end

% check for circular orbit

if (eccm > 0.00000001)
   argper = modulo(atan3(h, xk) - raan);
else
   argper = 0;
end

% true anomaly

tanom = modulo(xlambdat - raan - argper);

% load orbital element vector

oev(1) = sma;
oev(2) = eccm;
oev(3) = inc;
oev(4) = argper;
oev(5) = raan;
oev(6) = tanom;
```

```
function [month, day, year, hour, min, sec] = gdate2(jdate)

% convert Julian date to Gregorian (calendar) date

% input

%  jdate = julian day

% output

%  month = calendar month [1 - 12]
%  day   = calendar day [1 - 31]
%  year  = calendar year [yyyy]

%  note: day may include fractional part

% Orbital Mechanics MATLAB Toolbox

% Modifications made by Aries Broadnax
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

jd = jdate;

z = fix(jd + .5);
fday = jd + .5 - z;

if (fday < 0)
   fday = fday + 1;
   z = z - 1;
end

if (z < 2299161)
   a = z;
else
   alpha = floor((z - 1867216.25) / 36524.25);
   a = z + 1 + alpha - floor(alpha / 4);
end

b = a + 1524;
c = fix((b - 122.1) / 365.25);
d = fix(365.25 * c);
e = fix((b - d) / 30.6001);
day = b - d - fix(30.6001 * e) + fday;

if (e < 14)
   month = e - 1;
else
   month = e - 13;
end

if (month > 2)
   year = c - 4716;
else
   year = c - 4715;
end
```

```
hour =(mod(day,1) * 24);

min = (mod(hour,1) * 60);

sec = (mod(min,1) * 60);


hour = hour - mod(hour,1);

min = min - mod(min,1);

day = day - mod(day,1);
```

****

```
% Cowell Equations of motion for State Transition Matrix Propagation

function xdot = j2stm(t,x)

global mu req j2 c2 c3

xdot = zeros(42,1);

r = sqrt(x(1)*x(1) + x(2)*x(2) + x(3)*x(3));

a11=0; a12=0; a13=0; a14=1; a15=0; a16=0;
a21=0; a22=0; a23=0; a24=0; a25=1; a26=0;
a31=0; a32=0; a33=0; a34=0; a35=0; a36=1;
a44=0; a45=0; a46=0;
a54=0; a55=0; a56=0;
a64=0; a65=0; a66=0;

a41 = - (mu/(r^3)) + (3*x(1)*x(1)*mu/(r^5)) - (1.5*c2/(r^5)) ...
      + (7.5*c2*x(1)*x(1)/(r^7)) + (7.5*c2*x(3)*x(3)/(r^7)) - (52.5*c2*x(1)*x(1)*x(3)*x(3)/(r^9));
a42 = (3*mu*x(1)*x(2)/(r^5)) + (7.5*c2*x(1)*x(2)/(r^7)) - (52.5*c2*x(1)*x(2)*x(3)*x(3)/(r^9));
a43 = (3*mu*x(1)*x(3)/(r^5)) + (22.5*c2*x(1)*x(3)/(r^7)) - (52.5*c2*x(1)*x(3)*x(3)*x(3)/(r^9));

a51 = (3*mu*x(2)*x(1)/(r^5)) + (7.5*c2*x(1)*x(2)/(r^7)) - (52.5*c2*x(1)*x(2)*x(3)*x(3)/(r^9));
a52 = - (mu/(r^3)) + (3*mu*x(2)*x(2)/(r^5)) - (1.5*c2/(r^5)) ...
        + (7.5*c2*x(2)*x(2)/(r^7)) + (7.5*c2*x(3)*x(3)/(r^7)) - (52.5*c2*x(2)*x(2)*x(3)*x(3)/(r^9));
a53 = (3*mu*x(2)*x(3)/(r^5)) + (22.5*c2*x(2)*x(3)/(r^7)) - (52.5*c2*x(2)*x(3)*x(3)*x(3)/(r^9));

a61 = (3*mu*x(1)*x(3)/(r^5)) + (27*c3*x(1)*x(3)/(r^8)) - (60*c3*x(1)*x(3)*x(3)*x(3)/(r^10));
a62 = (3*mu*x(2)*x(3)/(r^5)) + (27*c3*x(2)*x(3)/(r^8)) - (60*c3*x(2)*x(3)*x(3)*x(3)/(r^10));
a63 = - (mu/(r^3)) + (3*mu*x(3)*x(3)/(r^5)) - (4.5*c3/(r^6)) ...
      + (49.5*c3*x(3)*x(3)/(r^8)) - (60*c3*x(3)*x(3)*x(3)*x(3)/(r^10));

xdot(1) = x(4);
xdot(2) = x(5);
xdot(3) = x(6);
xdot(4) = - (mu*x(1)/(r^3)) - (1.5*c2*x(1)/(r^5)) + (7.5*c2*x(1)*x(3)*x(3)/(r^7));
xdot(5) = - (mu*x(2)/(r^3)) - (1.5*c2*x(2)/(r^5)) + (7.5*c2*x(2)*x(3)*x(3)/(r^7));
xdot(6) = - (mu*x(3)/(r^3)) - (4.5*c3*x(3)/(r^6)) + (7.5*c3*x(3)*x(3)*x(3)/(r^8));
xdot(7) = a11*x(7) + a12*x(13) + a13*x(19) + a14*x(25) + a15*x(31) + a16*x(37);
xdot(8) = a11*x(8) + a12*x(14) + a13*x(20) + a14*x(26) + a15*x(32) + a16*x(38);
xdot(9) = a11*x(9) + a12*x(15) + a13*x(21) + a14*x(27) + a15*x(33) + a16*x(39);
xdot(10) = a11*x(10) + a12*x(16) + a13*x(22) + a14*x(28) + a15*x(34) + a16*x(40);
xdot(11) = a11*x(11) + a12*x(17) + a13*x(23) + a14*x(29) + a15*x(35) + a16*x(41);
xdot(12) = a11*x(12) + a12*x(18) + a13*x(24) + a14*x(30) + a15*x(36) + a16*x(42);
xdot(13) = a21*x(7) + a22*x(13) + a23*x(19) + a24*x(25) + a25*x(31) + a26*x(37);
xdot(14) = a21*x(8) + a22*x(14) + a23*x(20) + a24*x(26) + a25*x(32) + a26*x(38);
xdot(15) = a21*x(9) + a22*x(15) + a23*x(21) + a24*x(27) + a25*x(33) + a26*x(39);
xdot(16) = a21*x(10) + a22*x(16) + a23*x(22) + a24*x(28) + a25*x(34) + a26*x(40);
xdot(17) = a21*x(11) + a22*x(17) + a23*x(23) + a24*x(29) + a25*x(35) + a26*x(41);
xdot(18) = a21*x(12) + a22*x(18) + a23*x(24) + a24*x(30) + a25*x(36) + a26*x(42);
xdot(19) = a31*x(7) + a32*x(13) + a33*x(19) + a34*x(25) + a35*x(31) + a36*x(37);
xdot(20) = a31*x(8) + a32*x(14) + a33*x(20) + a34*x(26) + a35*x(32) + a36*x(38);
xdot(21) = a31*x(9) + a32*x(15) + a33*x(21) + a34*x(27) + a35*x(33) + a36*x(39);
xdot(22) = a31*x(10) + a32*x(16) + a33*x(22) + a34*x(28) + a35*x(34) + a36*x(40);
xdot(23) = a31*x(11) + a32*x(17) + a33*x(23) + a34*x(29) + a35*x(35) + a36*x(41);
xdot(24) = a31*x(12) + a32*x(18) + a33*x(24) + a34*x(30) + a35*x(36) + a36*x(42);
xdot(25) = a41*x(7) + a42*x(13) + a43*x(19) + a44*x(25) + a45*x(31) + a46*x(37);
xdot(26) = a41*x(8) + a42*x(14) + a43*x(20) + a44*x(26) + a45*x(32) + a46*x(38);
xdot(27) = a41*x(9) + a42*x(15) + a43*x(21) + a44*x(27) + a45*x(33) + a46*x(39);
xdot(28) = a41*x(10) + a42*x(16) + a43*x(22) + a44*x(28) + a45*x(34) + a46*x(40);
xdot(29) = a41*x(11) + a42*x(17) + a43*x(23) + a44*x(29) + a45*x(35) + a46*x(41);
xdot(30) = a41*x(12) + a42*x(18) + a43*x(24) + a44*x(30) + a45*x(36) + a46*x(42);
xdot(31) = a51*x(7) + a52*x(13) + a53*x(19) + a54*x(25) + a55*x(31) + a56*x(37);
xdot(32) = a51*x(8) + a52*x(14) + a53*x(20) + a54*x(26) + a55*x(32) + a56*x(38);
xdot(33) = a51*x(9) + a52*x(15) + a53*x(21) + a54*x(27) + a55*x(33) + a56*x(39);
xdot(34) = a51*x(10) + a52*x(16) + a53*x(22) + a54*x(28) + a55*x(34) + a56*x(40);
xdot(35) = a51*x(11) + a52*x(17) + a53*x(23) + a54*x(29) + a55*x(35) + a56*x(41);
xdot(36) = a51*x(12) + a52*x(18) + a53*x(24) + a54*x(30) + a55*x(36) + a56*x(42);
xdot(37) = a61*x(7) + a62*x(13) + a63*x(19) + a64*x(25) + a65*x(31) + a66*x(37);
xdot(38) = a61*x(8) + a62*x(14) + a63*x(20) + a64*x(26) + a65*x(32) + a66*x(38);
xdot(39) = a61*x(9) + a62*x(15) + a63*x(21) + a64*x(27) + a65*x(33) + a66*x(39);
xdot(40) = a61*x(10) + a62*x(16) + a63*x(22) + a64*x(28) + a65*x(34) + a66*x(40);
xdot(41) = a61*x(11) + a62*x(17) + a63*x(23) + a64*x(29) + a65*x(35) + a66*x(41);
```

```
xdot(42) = a61*x(12) + a62*x(18) + a63*x(24) + a64*x(30) + a65*x(36) + a66*x(42);

*************************************************************************************************************************

function jdate = julian (month, day, year)

% Julian date

% Input

%  month = calendar month [1 - 12]
%  day   = calendar day [1 - 31]
%  year  = calendar year [yyyy]

% Output

%  jdate = Julian date

% special notes

%  (1) calendar year must include all digits

%  (2) will report October 5, 1582 to October 14, 1582
%      as invalid calendar dates and stop

% Orbital Mechanics MATLAB Toolbox

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y = year;
m = month;
b = 0;
c = 0;

if (m <= 2)
   y = y - 1;
   m = m + 12;
end

if (y < 0)
   c = -.75;
end

% check for valid calendar date

if (year < 1582)
   % null
elseif (year > 1582)
   a = fix(y / 100);
   b = 2 - a + floor(a / 4);
elseif (month < 10)
   % null
elseif (month > 10)
   a = fix(y / 100);
   b = 2 - a + floor(a / 4);
elseif (day <= 4)
   % null
elseif (day > 14)
   a = fix(y / 100);
   b = 2 - a + floor(a / 4);
else
   clc; home;

   fprintf('\n\n  this is an invalid calendar date!!\n');

   keycheck;

   return;
end

jd = fix(365.25 * y + c) + fix(30.6001 * (m + 1));

jdate = jd + day + b + 1720994.5;

*************************************************************************************************************************

% search.m
% Search for the closest approach between to earth orbiting objects.

% Clear all stored variables
clear all

% Format the output to fixed point format with 15 digits
format long

% Initialize global variables
global jdate options

% Read in the spacecraft and debris orbit data
catdata
```

```
% Load various constants
constants

% Calculatte spacecraft and debris orbital elements
pos1 = [orb1(1) orb1(2) orb1(3)];
vel1 = [orb1(4) orb1(5) orb1(6)];
elements1 = eci2orb1(pos1,vel1);
semi1 = elements1(1);
eccen1 = elements1(2);
inc1 = elements1(3);
argu1 = elements1(4);
raan1 = elements1(5);
period1 = 2*pi*sqrt(semi1*semi1*semi1/mu);

pos2 = [orb2(1) orb2(2) orb2(3)];
vel2 = [orb2(4) orb2(5) orb2(6)];
elements2 = eci2orb1(pos2,vel2);
semi2 = elements2(1);
eccen2 = elements2(2);
inc2 = elements2(3);
argu2 = elements2(4);
raan2 = elements2(5);
period2 = 2*pi*sqrt(semi2*semi2*semi2/mu);

% Enter close approach search data
fprintf('\nEnter the close approach search duration (hours).\n');
hours = input('Number of hours = ');
nsecs = hours * 3600;

fprintf('\nEnter the minimum separation constraint (km).\n');
threat = input('Minimum separation constraint = ');

% Propagate state vectors to close approach search start date and time

jdate1 = julian(month1,day1,year1) + hour1/24 + min1/1440 + sec1/86400;
jdate = jdate1;
pt1 = (sdate - jdate1) * 86400;
ts = [0 pt1];
[sct, sc] = ode45('cow',ts,orb1,options);
[sca, scb] = size(sc);
scorb = [sc(sca,1) sc(sca,2) sc(sca,3) sc(sca,4) sc(sca,5) sc(sca,6)];
scorbit = scorb;

jdate2 = julian(month2,day2,year2) + hour2/24 + min2/1440 + sec2/86400;
jdate = jdate2;
pt2 = (sdate - jdate2) * 86400;
ts = [0 pt2];
[debt, deb] = ode45('cow',ts,orb2,options);
[deba, debb] = size(deb);
debris = [deb(deba,1) deb(deba,2) deb(deba,3) deb(deba,4) deb(deba,5) deb(deba,6)];
dborbit = debris;

% Calculate Close Approach

jdate = sdate;
[sct,sc] = rk4fixed('cow',[0 nsecs],scorb,round(nsecs)/10,0);
[dbt,db] = rk4fixed('cow',[0 nsecs],debris,round(nsecs)/10,0);
[a,b] = size(sc);
[c,d] = size(db);
for i = 1:a
   scpos = [sc(i,1) sc(i,2) sc(i,3)];
   dbpos = [db(i,1) db(i,2) db(i,3)];
   diff = dbpos - scpos;
   diffm(i) = sqrt(dot(diff,diff));
end
[val pos] = min(diffm);
ntime = sct(pos-1);
time1 = ntime;
span = sct(pos+1) - sct(pos-1);
ts = [0 ntime];
[sct, sc] = ode45('cow',ts,scorb,options);
[sca, scb] = size(sc);
scorb = [sc(sca,1) sc(sca,2) sc(sca,3) sc(sca,4) sc(sca,5) sc(sca,6)];
[debt, deb] = ode45('cow',ts,debris,options);
[deba, debb] = size(deb);
debris = [deb(deba,1) deb(deba,2) deb(deba,3) deb(deba,4) deb(deba,5) deb(deba,6)];
jdate = sdate + ntime/86400;

sdate2 = jdate;
[sct,sc] = rk4fixed('cow',[0 span],scorb,100,0);
[dbt,db] = rk4fixed('cow',[0 span],debris,100,0);
[a,b] = size(sc);
[c,d] = size(db);
clear diffm
for i = 1:a
   scpos = [sc(i,1) sc(i,2) sc(i,3)];
   dbpos = [db(i,1) db(i,2) db(i,3)];
   diff = dbpos - scpos;
   diffm(i) = sqrt(dot(diff,diff));
```

```
end
[val pos] = min(diffm);
ntime = sct(pos-1);
time2 = ntime;
span = sct(pos+1) - sct(pos-1);
ts = [0 ntime];
[sct, sc] = ode45('cow',ts,scorb,options);
[sca, scb] = size(sc);
scorb = [sc(sca,1) sc(sca,2) sc(sca,3) sc(sca,4) sc(sca,5) sc(sca,6)];
[debt, deb] = ode45('cow',ts,debris,options);
[deba, debb] = size(deb);
debris = [deb(deba,1) deb(deba,2) deb(deba,3) deb(deba,4) deb(deba,5) deb(deba,6)];
jdate = sdate2 + ntime/86400;

sdate3 = jdate;
[sct,sc] = rk4fixed('cow',[0 span],scorb,300,0);
[dbt,db] = rk4fixed('cow',[0 span],debris,300,0);
[a,b] = size(sc);
[c,d] = size(db);
time = sct;
clear diffm
for i = 1:a
   scpos = [sc(i,1) sc(i,2) sc(i,3)];
   dbpos = [db(i,1) db(i,2) db(i,3)];
   diff = dbpos - scpos;
   diffm(i) = sqrt(dot(diff,diff));
end
[val pos] = min(diffm);
ntime = sct(pos-1);
time3 = ntime;
span = sct(pos+1) - sct(pos-1);
ts = [0 ntime];
[sct, sc] = ode45('cow',ts,scorb,options);
[sca, scb] = size(sc);
scorb = [sc(sca,1) sc(sca,2) sc(sca,3) sc(sca,4) sc(sca,5) sc(sca,6)];
[debt, deb] = ode45('cow',ts,debris,options);
[deba, debb] = size(deb);
debris = [deb(deba,1) deb(deba,2) deb(deba,3) deb(deba,4) deb(deba,5) deb(deba,6)];
coltime = time1 + time2 + time3; %seconds from common point
jdate = sdate3 + ntime/86400;
coldate = jdate;

violation = 0;
for i = 1:a
   if (diffm(i) < 2.0) & (violation == 0)
      violation = 1;
      plus = time(i);
      bdate = sdate3 + (plus/86400);
      [bmon, bday, byr, bhr, bmin, bsec] = gdate2(bdate);
      [dstr,tstr] = strdate(bday, bmon, byr, bhr, bmin, bsec);
      fprintf('\nMinimum Separation Distance Violation Begins\n');
      fprintf('%s  km @  %s  %s\n',diffm(i),dstr,tstr);
   end
   if (diffm(i) > 2.0) & (violation == 1)
      add = time(i);
      edate = sdate3 + (add/86400);
          [emon, eday, eyr, ehr, emin, esec] = gdate2(edate);
      [dstr,tstr] = strdate(eday, emon, eyr, ehr, emin, esec);
      fprintf('\nMinimum Separation Distance Violation Ends\n');
      fprintf('%s  km @  %s  %s\n',diffm(i),dstr,tstr);
      violation = 2;
   end
   if (i == a) & (violation == 0)
      fprintf('\nThere was no minimum separation distance violation.\n');
   end
   if (i == a) & (violation == 1)
     fprintf('\nSearch ended inside violation deadband. Choose a longer search duration.\n\n');
   end
end

[cmon, cday, cyr, chr, cmin, csec] = gdate2(coldate);
[dstr,tstr] = strdate(cday, cmon, cyr, chr, cmin, csec);
fprintf('\nActual Minimum Separation Distance\n');
fprintf('%s  km @  %s  %s\n',diffm(pos),dstr,tstr);

sccolvec = scorb
dbcolvec = debris

*****************************************************************************************************************************

function [rasc, decl, rmoon] = moon (jdate)

% lunar ephemeris

% input

%  jdate = julian date

% output
```

```
% rasc  = right ascension of the moon (radians)
%          (0 <= rasc <= 2 pi)
% decl  = declination of the moon (radians)
%          (-pi/2 <= decl <= pi/2)
% rmoon = eci position vector of the moon (km)

% note

%  coordinates are inertial, geocentric,
%  equatorial and true-of-date

% Orbital Mechanics MATLAB Toolbox

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

atr = pi / 648000;

rmoon = zeros(3, 1);

% time arguments

djd = jdate - 2451545;

t = (djd / 36525) + 1;

% fundamental arguments (radians)

qm = r2r(0.374897 + 0.03629164709 * djd);
qm2 = 2 * qm;
qm3 = 3 * qm;
fm = r2r(0.259091 + 0.0367481952 * djd);
fm2 = 2 * fm;
em = r2r(0.827362 + 0.03386319198 * djd);
em2 = 2 * em;
em4 = 4 * em;
qs = r2r(0.993126 + 0.0027377785 * djd);
lv = r2r(0.505498 + 0.00445046867 * djd);
lm = r2r(0.606434 + 0.03660110129 * djd);
ls = r2r(0.779072 + 0.00273790931 * djd);
rm = r2r(0.347343 - 0.00014709391 * djd);

% geocentric, ecliptic longitude of the moon (radians)

l = 22640 * sin(qm) - 4586 * sin(qm - em2) + 2370 * sin(em2);
l = l + 769 * sin(qm2) - 668 * sin(qs) - 412 * sin(fm2);
l = l - 212 * sin(qm2 - em2) - 206 * sin(qm - em2 + qs);
l = l + 192 * sin(qm + em2) + 165 * sin(em2 - qs);
l = l + 148 * sin(qm - qs) - 125 * sin(em) - 110 * sin(qm + qs);
l = l - 55 * sin(fm2 - em2) - 45 * sin(qm + fm2) + 40 * sin(qm - fm2);
l = l - 38 * sin(qm - em4) + 36 * sin(qm3) - 31 * sin(qm2 - em4);
l = l + 28 * sin(qm - em2 - qs) - 24 * sin(em2 + qs) + 19 * sin(qm - em);
l = l + 18 * sin(em + qs) + 15 * sin(qm + em2 - qs) + 14 * sin(qm2 + em2);
l = l + 14 * sin(em4) - 13 * sin(qm3 - em2) - 17 * sin(rm);
l = l - 11 * sin(qm + 16 * ls - 18 * lv) + 10 * sin(qm2 - qs) ...
    + 9 * sin(qm - fm2 - em2);
l = l + 9 * (cos(qm + 16 * ls - 18 * lv) - sin(qm2 - em2 + qs)) ...
    - 8 * sin(qm + em);
l = l + 8 * (sin(2 * (em - qs)) - sin(qm2 + qs)) - 7 * (sin(2 * qs) ...
    + sin(qm - 2 * (em - qs)) - sin(rm));
l = l - 6 * (sin(qm - fm2 + em2) + sin(fm2 + em2)) ...
    - 4 * (sin(qm - em4 + qs) - t * cos(qm + 16 * ls - 18 * lv));
l = l - 4 * (sin(qm2 + fm2) - t * sin(qm + 16 * ls - 18 * lv));
l = l + 3 * (sin(qm - 3 * em) - sin(qm + em2 + qs) ...
    - sin(qm2 - em4 + qs) + sin(qm - 2 * qs) + sin(qm - em2 - 2 * qs));
l = l - 2 * (sin(qm2 - em2 - qs) + sin(fm2 - em2 + qs) - sin(qm + em4));
l = l + 2 * (sin(4 * qm) + sin(em4 - qs) + sin(qm2 - em));

plon = lm + atr * l;

% geocentric, ecliptic latitude of the moon (radians)

b = 18461 * sin(fm) + 1010 * sin(qm + fm) + 1000 * sin(qm - fm);
b = b - 624 * sin(fm - em2) - 199 * sin(qm - fm - em2) ...
    - 167 * sin(qm + fm - em2);
b = b + 117 * sin(fm + em2) + 62 * sin(qm2 + fm) + 33 * sin(qm - fm + em2);
b = b + 32 * sin(qm2 - fm) - 30 * sin(fm - em2 + qs) ...
    - 16 * sin(qm2 - em2 + fm);
b = b + 15 * sin(qm + fm + em2) + 12 * sin(fm - em2 - qs) ...
    - 9 * sin(qm - fm - em2 + qs);
b = b - 8 * (sin(fm + rm) - sin(fm + em2 - qs)) ...
    - 7 * sin(qm + fm - em2 + qs);
b = b + 7 * (sin(qm + fm - qs) - sin(qm - fm - em4));
b = b - 6 * (sin(fm + qs) + sin(3 * fm) - sin(qm - fm - qs));
b = b - 5 * (sin(fm + em) + sin(qm + fm + qs) + sin(qm - fm + qs) ...
    - sin(fm - qs) - sin(fm - em));
b = b + 4 * (sin(qm3 + fm) - sin(fm - em4)) - 3 * (sin(qm - fm - em4) ...
    - sin(qm - 3 * fm));
b = b - 2 * (sin(qm2 - fm - em4) + sin(3 * fm - em2) - sin(qm2 - fm + em2) ...
    - sin(qm - fm + em2 - qs));
```

```
plat = atr * (b + 2 * (sin(gm2 - fm - em2) + sin(gm3 - fm)));

% obliquity of the ecliptic (radians)

obliq = atr * (84428 - 47 * t + 9 * cos(rm));

% geocentric distance (kilometers)

r = 60.36298 - 3.27746 * cos(gm) - .57994 * cos(gm - em2);
r = r - .46357 * cos(em2) - .08904 * cos(gm2) + .03865 * cos(gm2 - em2);
r = r - .03237 * cos(em2 - gs) - .02688 * cos(gm + em2) ...
    - .02358 * cos(gm - em2 + gs);
r = r - .0203 * cos(gm - gs) + .01719 * cos(em) + .01671 * cos(gm + gs);
r = r + .01247 * cos(gm - fm2) + .00704 * cos(gs) + .00529 * cos(em2 + gs);
r = r - .00524 * cos(gm - em4) + .00398 * cos(gm - em2 - gs) ...
    - .00366 * cos(gm3);
r = r - .00295 * cos(gm2 - em4) - .00263 * cos(em + gs) ...
    + .00249 * cos(gm3 - em2);
r = r - .00221 * cos(gm + em2 - gs) + .00185 * cos(fm2 - em2) ...
    - .00161 * cos(2 * (em - gs));
r = r + 0.00147 * cos(gm + fm2 - em2) - 0.00142 * cos(em4) ...
    + 0.00139 * cos(gm2 - em2 + gs);

rmm = 6378.14 * (r - 0.00118 * cos(gm - em4 + gs) - 0.00116 * cos(gm2 + em2) ...
      - 0.0011 * cos(gm2 - gs));

% geocentric, equatorial right ascension and declination (radians)

a = sin(plon) * cos(obliq) - tan(plat) * sin(obliq);
b = cos(plon);

rasc = atan3(a, b);

decl = asin(sin(plat) * cos(obliq) + cos(plat) * sin(obliq) * sin(plon));

% geocentric position vector of the moon (kilometers)

rmoon(1) = rmm * cos(rasc) * cos(decl);
rmoon(2) = rmm * sin(rasc) * cos(decl);
rmoon(3) = rmm * sin(decl);
```

```
function [pv1, vv1, pv2, vv2] = piter(r1, r2, tof)

global mu

mr1=sqrt(dot(r1,r1));
mr2=sqrt(dot(r2,r2));
cp = cross(r1,r2);
cp3 = cp(3);
sinta = sqrt(dot(cp,cp))/(mr1*mr2);
costa = dot(r1,r2)/(mr1*mr2);
ta1 = atan2(sinta,costa);

if cp3 < 0;
   ta = 2*pi - ta1;
   sinta = sin(ta);
   costa = cos(ta);
else
   ta = ta1;
end

k = mr1*mr2*(1-cos(ta));
l = mr1+mr2;
m = mr1*mr2*(1+cos(ta));

p1 = k/(l+sqrt(2*m));
p2 = k/(l-sqrt(2*m));

p = (p1 + p2)/2;
pdiff = 1;

while abs(pdiff) > 1.0e-10
a = (m*k*p)/((2*m-l*l)*p*p+2*k*l*p-k*k);
f = 1-(mr2/p)*(1-costa);
g = mr1*mr2*sin(ta)/(sqrt(mu*p));
fd = sqrt(mu/p)*tan(ta/2)*(((1-costa)/p)-(1/mr1)-(1/mr2));
gd = 1-(mr1/p)*(1-costa);
if a >= 0
   cosde = 1-(mr1/a)*(1-f);
   sinde = -mr1*mr2*fd/sqrt(mu*a);
   de = atan2(sinde,cosde);
   if de < 0
      de = 2*pi + de;
   end
   t = g+sqrt(a*a*a/mu)*(de-sinde);
   dtdp = (-g/(2*p))-(3/2)*a*(t-g)*((k*k+(2*m-l*l)*p*p)/(m*k*p*p))+sqrt((a*a*a)/mu)*((2*k*sinde)/(p*(k-l*p)));
   pnew = p + (tof-t)/dtdp;
end
```

```
if a < 0
   df = acosh(1-(mr1/a)*(1-f));
   t = g+sqrt(((-a)^3)/mu)*(sinh(df)-df);
   dtdp = (-g/(2*p))-(3/2)*a*(t-g)*((k*k+(2*m-1*l)*p*p)/(m*k*p*p))-sqrt((-a^3)/mu)*((2*k*sinh(df))/(p*(k-1*p)));
   pnew = p + (tof-t)/dtdp;
end
pold = p;
p = pnew;
pdiff = pnew-pold;
end

v1 = (r2-f*r1)/g;
v2 = fd*r1+gd*v1;
cr2 = f*r1+g*v1;
checkr2 = abs(sqrt(dot(cr2,cr2))-sqrt(dot(r2,r2)));
if checkr2 > .001
   disp('Check for r2 failed')
end
pv1 = r1;
vv1 = v1;
pv2 = r2;
vv2 = v2;
```

```
function [vel] = plamb(vec,t,endp);

global options


pos = [vec(1) vec(2) vec(3)];
vel = [vec(4) vec(5) vec(6)];

vec = [pos vel ...
       1 0 0 0 0 0 ...
       0 1 0 0 0 0 ...
       0 0 1 0 0 0 ...
       0 0 0 1 0 0 ...
       0 0 0 0 1 0 ...
       0 0 0 0 0 1];

drss = 1;
pmvec = [pos vel];
while drss > 1.0e-10
   [ti,x] = ode45('j2stm',[0 t],vec,options);
   stm1 = getstm(x);
     for j = 1:1:3
          for k = 1:1:3
          stm12(j,k) = stm1(j,k+3);
          end
     end
   [tout,xout] = ode45('cow',[0 t],pmvec,options);
   [aa,bb] = size(xout);
   xpos = [xout(aa,1) xout(aa,2) xout(aa,3)];
   drf = endp - xpos;
   drss = sqrt(dot(drf,drf));
   vc = inv(stm12)*drf';
   vel = vel + vc';
   pmvec = [pos vel];
end
```

```
function [rasc, decl, rsun] = sun (jdate)

% solar ephemeris

% input

%  jdate = julian day

% output

%  rasc = right ascension of the sun (radians)
%         (0 <= rasc <= 2 pi)
%  decl = declination of the sun (radians)
%         (-pi/2 <= decl <= pi/2)
%  rsun = eci position vector of the sun (km)

% note

%  coordinates are inertial, geocentric,
%  equatorial and true-of-date

% Orbital Mechanics MATLAB Toolbox

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
str = pi / 648000;

% time arguments

djd = jdate - 2451545;

t = (djd / 36525) + 1;

% fundamental arguments (radians)

gs = r2r(0.993126 + 0.0027377785 * djd);
lm = r2r(0.606434 + 0.03660110129 * djd);
ls = r2r(0.779072 + 0.00273790931 * djd);
g2 = r2r(0.140023 + 0.00445036173 * djd);
g4 = r2r(0.053856 + 0.00145561327 * djd);
g5 = r2r(0.056531 + 0.00023080893 * djd);
rm = r2r(0.347343 - 0.00014709391 * djd);

% geocentric, ecliptic longitude of the sun (radians)

plon = 6910 * sin(gs) + 72 * sin(2 * gs) - 17 * t * sin(gs);
plon = plon - 7 * cos(gs - g5) + 6 * sin(lm - ls) ...
       + 5 * sin(4 * gs - 8 * g4 + 3 * g5);
plon = plon - 5 * cos(2 * (gs - g2)) - 4 * (sin(gs - g2) ...
       - cos(4 * gs - 8 * g4 + 3 * g5));
plon = plon + 3 * (sin(2 * (gs - g2)) - sin(g5) - sin(2 * (gs - g5)));
plon = ls + str * (plon - 17 * sin(rm));

% geocentric distance of the sun (kilometers)

rsm = 149597870.66 * (1.00014 - 0.01675 * cos(gs) - 0.00014 * cos(2 * gs));

% obliquity of the ecliptic (radians)

obliq = str * (84428 - 47 * t + 9 * cos(rm));

% geocentric, equatorial right ascension and declination (radians)

a = sin(plon) * cos(obliq);
b = cos(plon);

rasc = atan3(a, b);
decl = asin(sin(obliq) * sin(plon));

% geocentric position vector of the sun (kilometers)

rsun(1) = rsm * cos(rasc) * cos(decl);
rsun(2) = rsm * sin(rasc) * cos(decl);
rsun(3) = rsm * sin(decl);



function [pvec,tvec] = pvhist(tof, r, v, pvo,pvdo);

global options

n = 1;

pvec(n) = sqrt(dot(pvo,pvo));
tvec(n) = 0;

n = 2;

vec = [r v ...
       1 0 0 0 0 0 ...
       0 1 0 0 0 0 ...
       0 0 1 0 0 0 ...
       0 0 0 1 0 0 ...
       0 0 0 0 1 0 ...
       0 0 0 0 0 1];

for i = 60:60:tof
   [t,x] = ode45('j2stm',[0 i],vec,options);
   stm = getstm(x);
   for k = 1:1:3;
      for m = 1:1:3;
         p11(k,m) = stm(k,m);
         p12(k,m) = stm(k,3+m);
         %p21(k,m) = stm(k+3,m);
         %p22(k,m) = stm(k+3,m+3);
      end
   end
   pv = p11*pvo' + p12*pvdo;
   pvec(n) = sqrt(dot(pv,pv));
   tvec(n) = i;
   n = n + 1;
end
```

```
function [pvdi] = pvinit(tof, r1, v1, dv1, dv2)

global options

% primer vector initialization

% required by primer.m

% Orbital Mechanics MATLAB Toolbox

% Modifications made by Aries Broadnax

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% compute primer vector at first impulse

dv1m = sqrt(dot(dv1, dv1));

pvi = dv1 / dv1m;

% compute primer vector at second impulse

dv2m = sqrt(dot(dv2, dv2));

pvf = dv2 / dv2m;

 % compute initial value of primer derivative

 vec = [r1 v1 ...
       1 0 0 0 0 0 ...
       0 1 0 0 0 0 ...
       0 0 1 0 0 0 ...
       0 0 0 1 0 0 ...
       0 0 0 0 1 0 ...
       0 0 0 0 0 1];

[t,x] = ode45('j2stm',[0 tof],vec,options);

strm = getstm(x);

for i = 1:1:3
    for j = 1:1:3
        stm11(i, j) = strm(i, j);
        stm12(i, j) = strm(i, j + 3);
    end
end

tvec1 = stm11 * pvi';

tvec2 = pvf' - tvec1;

stm12i = inv(stm12);

pvdi = stm12i * tvec2;

************************************************************************************************************************

function [tout,xout] = rk4fixed(FUN,tspan,x0,Nsteps,ode_fcn_format,trace,count)

% Copyright (C) 2001, 2000 Marc Compere
% This file is intended for use with Octave.
% rk4fixed.m is free software; you can redistribute it and/or modify it
% under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 2, or (at your option)
% any later version.
%
% rk4fixed.m is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
% General Public License for more details at www.gnu.org/copyleft/gpl.html.
%
% -------------------------------------------------------------------
%
% rk4fixed (v1.11) is a 4th order Runge-Kutta numerical integration routine.
% It requires 4 function evaluations per step.
%
% Usage:
%         [tout, xout] = rk4fixed(FUN, tspan, x0, Nsteps, ode_fcn_format, trace, count)
%
% INPUT:
% FUN    - String containing name of user-supplied problem derivatives.
%          Call: xprime = fun(t,x) where FUN = 'fun'.
%          t      - Time or independent variable (scalar).
%          x      - Solution column-vector.
%          xprime - Returned derivative COLUMN-vector; xprime(i) = dx(i)/dt.
% tspan  - [ tstart, tfinal ]
% x0     - Initial value COLUMN-vector.
% Nsteps - number of steps used to span [ tstart, tfinal ]
% ode_fcn_format - this specifies if the user-defined ode function is in
```

```
%          the form:      xprime = fun(t,x)    (ode_fcn_format=0, default)
%          or:            xprime = fun(x,t)    (ode_fcn_format=1)
%          Matlab's solvers comply with ode_fcn_format=0 while
%          Octave's lsode() and sdirk4() solvers comply with ode_fcn_format=1.
% trace  - If nonzero, each step is printed. (optional, default: trace = 0).
% count  - if nonzero, variable 'rhs_counter' is initalized, made global
%          and counts the number of state-dot function evaluations
%          'rhs_counter' is incremented in here, not in the state-dot file
%          simply make 'rhs_counter' global in the file that calls rk4fixed
%
% OUTPUT:
% tout   - Returned integration time points (row-vector).
% xout   - Returned solution, one solution column-vector per tout-value.
%
% The result can be displayed by: plot(tout, xout).
%
% Marc Compere
% CompereM@asme.org
% created : 06 October 1999
% modified: 17 January 2001

if nargin < 7, count = 0; end
if nargin < 6, trace = 0; end
if nargin < 5, Nsteps = 100/(tspan(2)-tspan(1)); end % <-- 100 is a guess for a default,
                                                     %  try verifying the solution with rk8fixed
if nargin < 4, ode_fcn_format = 0; end

if count==1,
 global rhs_counter
 if ~exist('rhs_counter'),rhs_counter=0;,end
end % if count

% Initialization
t = tspan(1);
h = (tspan(2)-tspan(1))/Nsteps;
xout = x0;
tout(1) = t;
x = x0';
halfh = 0.5*h;

if trace
 clc, t, h, x
end

for i=1:Nsteps,
     if (ode_fcn_format==0),
      RK1 = feval(FUN,t,x);
      thalf = t+halfh;
      xtemp = x+halfh*RK1;
      RK2 = feval(FUN,thalf,xtemp);
      xtemp = x+halfh*RK2;
      RK3 = feval(FUN,thalf,xtemp);
      tfull = t+h;
      xtemp = x+h*RK3;
      RK4 = feval(FUN,tfull,xtemp);
     else,
      RK1 = feval(FUN,x,t);
      thalf = t+halfh;
      xtemp = x+halfh*RK1;
      RK2 = feval(FUN,xtemp,thalf);
      xtemp = x+halfh*RK2;
      RK3 = feval(FUN,xtemp,thalf);
      tfull = t+h;
      xtemp = x+h*RK3;
      RK4 = feval(FUN,xtemp,tfull);
     end % if (ode_fcn_format==0)

     % increment rhs_counter
     if count==1,
      rhs_counter = rhs_counter + 4;
     end % if

     t = t + h;
     x = (x+h/6*(RK1+2.0*(RK2+RK3)+RK4));
     tout = [tout; t];
     xout = [xout; x.'];
     if trace,
      home, t, h, x
     end

end
```

```
% spmans.m
% Calculate single-impulse evasive maneuvers

% Read in the desired maneuver time
fprintf('\nHow long before the minimum separation point');
fprintf('\nwould you like to perform the avoidance maneuver.\n\n');
```

```
fprintf('Enter the [hours minutes secs], include the brackets and spaces.\n');
fprintf('i.e.  2 hours, 30 mins, and 30 secs = [2 30 30]\n');

prior = input('==>  ');
secbef = prior(1)* 3600 + prior(2)* 60 + prior(3);

timespan = [0 coltime-secbef];

% Determin the position of thee spacecraft and the debris at the desired maneuver time
jdate = sdate;
[sbackt, sbackv] = ode45('cow',timespan,scorbit,options);
[sba, sbb] = size(sbackv);
sstartpos = [sbackv(sba,1) sbackv(sba,2) sbackv(sba,3)];
sstartvel = [sbackv(sba,4) sbackv(sba,5) sbackv(sba,6)];

[dbackt, dbackv] = ode45('cow',timespan,dborbit,options);
[dba, dbb] = size(dbackv);
dstartpos = [dbackv(dba,1) dbackv(dba,2) dbackv(dba,3)];
dstartvel = [dbackv(dba,4) dbackv(dba,5) dbackv(dba,6)];

dbcolpos = [dbcolvec(1) dbcolvec(2) dbcolvec(3)];

% Calculate the ECI avoidance circle locations, the delta v to maneuver to these locations
% and the change in the orbital elements after the maneuver is performed

n = 1;
for i = 0:45:360;
   ta = atan2(dbcolvec(2),dbcolvec(1));
   x = threat * cos(i*pi/180)* cos(ta);
   y = threat * cos(i*pi/180)* sin(ta);
   z = threat * sin(i*pi/180);
   threatvec = [x y z];
   avdvec = dbcolpos + threatvec;
   [pmstartpos, pmstartvel, endpos, endvel] = piter(sstartpos,avdvec,secbef);
   ts = [0 secbef];
   drss = 1;
   vel = pmstartvel;
   pmvec = [pmstartpos vel];
   while drss > 1.0e-10
   stml = stm(secbef,pmstartpos,vel);
   for j = 1:1:3
      for k = 1:1:3
         stm12(j,k) = stml(j,k+3);
      end
   end
   [tout,xout] = ode45('cow',ts,pmvec,options);
   [aa,bb] = size(xout);
   xpos = [xout(aa,1) xout(aa,2) xout(aa,3)];
   drf = endpos - xpos;
   drss = sqrt(dot(drf,drf));
   vc = inv(stm12)*drf';
   vel = vel + vc';
   pmvec = [pmstartpos vel];
   end
   pmstartvel = vel;
   classical = eci2orbl(pmstartpos,pmstartvel);
   semi = classical(1);
   delsemi(n) = (semi-semi1);
   eccen = classical(2);
   delecc(n) = (eccen-eccen1);
   incl = classical(3);
   delinc(n) = (incl-incl);
   argu = classical(4);
   delarg(n) = (argu-argu1);
   raan = classical(5);
   delraan(n) = (raan-raan1);
   dv = pmstartvel - sstartvel;
   dvm(n) = sqrt(dot(dv,dv));
   dvmps(n) = 1000*dvm(n);
   tfa(n) = i;
   n = n + 1;
   i
end

% Plot the results
subplot(3,2,1),plot(tfa,delsemi)
title('Change in the Semi Major Axis')
ylabel('Delta a (km)')
subplot(3,2,2),plot(tfa,delecc)
title('Change in the Eccentricity')
ylabel('Delta e')
subplot(3,2,3),plot(tfa,delinc*rtd)
title('Change in the Inclination')
ylabel('Delta i (deg)')
subplot(3,2,4),plot(tfa,delarg*rtd)
title('Change in the Argument of Perigee')
ylabel('Delta w (deg)')
subplot(3,2,5),plot(tfa,delraan*rtd)
title('Change in the RAAN')
xlabel('Avoidance Circle Location (deg)')
```

```
ylabel('Delta O (deg)')
subplot(3,2,6),plot(tfa,dvmps)
title('Required Change in Velocity')
xlabel('Avoidance Circle Location (deg)')
ylabel('Delta V (m/s)')

*****************************************************************************************************************

function [datestr,timestr] = strdate(d, m, y, h, n, s)

if d < 10
   d = num2str(d);
   d = strcat('0',d);
else
   d = num2str(d);
end

if m < 10
   m = num2str(m);
   m = strcat('0',m);
else
   m = num2str(m);
end

y = num2str(y);

if h < 10
   h = num2str(h);
   h = strcat('0',h);
else
   h = num2str(h);
end

if n < 10
   n = num2str(n);
   n = strcat('0',n);
else
   n = num2str(n);
end

if s < 10
   s = num2str(s);
   s = strcat('0',s);
else
   s = num2str(s);
end

datestr = strcat(d,'-',m,'-',y);
timestr = strcat(h,':',n,':',s);

*****************************************************************************************************************

%TRESPMANS Determine Optimal Three-Impulse evasive maneuvers

fprintf('\nHow long before the minimum separation point');
fprintf('\nwould you like to perform the avoidance maneuver.\n\n');
fprintf('Enter the [hours minutes secs], include the brackets and spaces.\n');
fprintf('i.e.  2 hours, 30 mins, and 30 secs = [2 30 30]\n');

prior = input('==>  ');
secbef = prior(1)* 3600 + prior(2)* 60 + prior(3);

fprintf('\nWhat position on the avoiadance circle');
fprintf('\nwill be the destination for the first impulse.\n\n');
fprintf('Enter the angle in degrees.\n');

acl = input('==>  ');

timespan = [0 coltime-secbef];

jdate = sdate;
[sbackt, sbackv] = ode45('ccw',timespan,scorbit,options);
[sba, sbb] = size(sbackv);
sstartpos = [sbackv(sba,1) sbackv(sba,2) sbackv(sba,3)];
sstartvel = [sbackv(sba,4) sbackv(sba,5) sbackv(sba,6)];
sstartvec = [sstartpos sstartvel];

[dbackt, dbackv] = ode45('ccw',timespan,dborbit,options);
[dba, dbb] = size(dbackv);
dstartpos = [dbackv(dba,1) dbackv(dba,2) dbackv(dba,3)];
dstartvel = [dbackv(dba,4) dbackv(dba,5) dbackv(dba,6)];
dstartvec = [dstartpos dstartvel];

sccolpos = [sccolvec(1) sccolvec(2) sccolvec(3)];
dbcolpos = [dbcolvec(1) dbcolvec(2) dbcolvec(3)];

ta = atan2(dbcolvec(2),dbcolvec(1));
x = threat * cos(acl*pi/180)* cos(ta);
y = threat * cos(acl*pi/180)* sin(ta);
z = threat * sin(acl*pi/180);
```

```
threatvec = [x y z];
avdvec = dbcolpos + threatvec;
[pmstartpos, pmstartvel, endpos, endvel] = piter(sstartpos,avdvec,secbef);
pmstartvec = [pmstartpos pmstartvel];
pmstartvel = plamb(pmstartvec,secbef,endpos);
pmstartvec = [pmstartpos pmstartvel];

timespan = [0 secbef];
[cat ca] = ode45('cow',timespan,pmstartvec,options);
[caa, cab] = size(ca);
endpos = [ca(caa,1) ca(caa,2) ca(caa,3)];
endvel = [ca(caa,4) ca(caa,5) ca(caa,6)];

dv1 = pmstartvel - sstartvel;
dv1m = sqrt(dot(dv1,dv1));
trajelem = eci2orbl(pmstartpos,pmstartvel);
sma = trajelem(1);
per = 2*pi*sqrt(sma*sma*sma/mu);

clear tdv dv2m dv3m rostime min
n = 1;
jdate = coldate;
for w = 4445:65:60:per
   ts = [0 w];
     [stationt, station] = ode45('cow',ts,sccolvec,options);
     [sta, stb] = size(station);
     stationpos = [station(sta,1) station(sta,2) station(sta,3)];
     stationvel = [station(sta,4) station(sta,5) station(sta,6)];
     [returnpos,returnvel,entrypos,entryvel] = piter(endpos,stationpos,w);
     returnvec = [returnpos returnvel];
   returnvel = plamb(returnvec,w,entrypos);
   returnvec = [returnpos returnvel];
   ts = [0 w];
     [ret re] = ode45('cow',ts,returnvec,options);
     [rea, reb] = size(re);
     entrypos = [re(rea,1) re(rea,2) re(rea,3)];
     entryvel = [re(rea,4) re(rea,5) re(rea,6)];
   dv2 = returnvel - endvel;
   dv2m(n) = sqrt(dot(dv2,dv2));
   dv3 = stationvel - entryvel;
   dv3m(n) = sqrt(dot(dv3,dv3));
   tdv(n) = dv1m + (dv2m(n) + dv3m(n));
   rostime(n) = w;
   n = n + 1;
end
[val,pos] = min(tdv);
rost = rostime(pos);
ts = [0 rost];
[stationt, station] = ode45('cow',ts,sccolvec,options);
[sta, stb] = size(station);
stationpos = [station(sta,1) station(sta,2) station(sta,3)];
stationvel = [station(sta,4) station(sta,5) station(sta,6)];
[returnpos,returnvel,entrypos,entryvel] = piter(endpos,stationpos,rost);
returnvec = [returnpos returnvel];
returnvel = plamb(returnvec,rost,entrypos);
returnvec = [returnpos returnvel];
[ret re] = ode45('cow',ts,returnvec,options);
[rea, reb] = size(re);
entrypos = [re(rea,1) re(rea,2) re(rea,3)];
entryvel = [re(rea,4) re(rea,5) re(rea,6)];
dv2 = returnvel - endvel;
dv2m = sqrt(dot(dv2,dv2));
pvi = dv2/dv2m;
dv3 = stationvel - entryvel;
dv3m = sqrt(dot(dv3,dv3));
pvf = dv3/dv3m;
total1 =  dv1m + dv2m + dv3m
pvdi = pvinit(rost, returnpos, returnvel, dv2, dv3);
[pv pvt] = pvhist(rost, returnpos, returnvel, pvi,pvdi);
%plot(pvt,pv)
%keyboard

% Additional Impulse
[val pos] = max(pv);
ait = pvt(pos);
to2tm = ait;
tm2tf = rost - ait;
[ait, ai] = ode45('cow',to2tm,returnvec,options);
[aia, aib] = size(ai);
aiposition = [ai(aia,1) ai(aia,2) ai(aia,3)];
vec = [returnpos returnvel 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1];
[t,x] = ode45('j2stm',[0 to2tm],vec,options);
stmm = stm(to2tm,returnpos,returnvel)
stmm = getstm(x)
pveci = [pvi';pvdi];
pvecm = stmm*pveci;
pposm = [pvecm(1) pvecm(2) pvecm(3)];
pposmm = sqrt(dot(pposm,pposm));
vec = [returnpos returnvel 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1];
[t,x] = ode45('j2stm',[0 to2tm],vec,options);
```

```
stmim = getstm(x);
for k = 1:1:3;
   for m = 1:1:3;
      p11im(k,m) = stmim(k,m);
      p12im(k,m) = stmim(k,3+m);
      p21im(k,m) = stmim(k+3,m);
      p22im(k,m) = stmim(k+3,m+3);
   end
end
vec = [entrypos entryvel 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1];
[t,x] = ode45('j2stm',[0 -tm2tf],vec,options);
stmfm = getstm(x);
for k = 1:1:3;
   for m = 1:1:3;
      p11fm(k,m) = stmfm(k,m);
      p12fm(k,m) = stmfm(k,3+m);
      p21fm(k,m) = stmfm(k+3,m);
      p22fm(k,m) = stmfm(k+3,m+3);
   end
end
biga = p22fm*inv(p12fm) - p22im*inv(p12im);
alpha = inv(p12im)*inv(biga)*(pposm/pposmm)';
beta = inv(p12fm)*inv(biga)*(pposm/pposmm)';
bdvf = dot(beta,dv3);
advo = dot(alpha,dv2);
aa = dot(alpha,alpha);
bb = dot(beta,beta);
num = (bdvf/dv3m) - (advo/dv2m) - 1;
den1 = (aa-((advo*advo)/(dv2m*dv2m)))/(dv2m);
den2 = (bb-((bdvf*bdvf)/(dv3m*dv3m)))/(dv3m);
lilc = num/(den1*den2);
delpos = inv(biga)*(pposm/pposmm)'*lilc;
aipos = aiposition + delpos';
[rtrj1pos,rtrj1vel,midc1pos,midc1vel] = piter(endpos,aipos,to2tm);
rtr1vec = [rtrj1pos rtrj1vel];
rtrj1vel = plamb(rtr1vec,to2tm,midc1pos);
rtr1vec = [rtrj1pos rtrj1vel];
[ret re] = ode45('cow',[0 to2tm],rtr1vec,options);
[rea, reb] = size(re);
midc1pos = [re(rea,1) re(rea,2) re(rea,3)];
midc1vel = [re(rea,4) re(rea,5) re(rea,6)];
[rtrj2pos,rtrj2vel,rospos,rosvel] = piter(aipos,stationpos,tm2tf);
rtr2vec = [rtrj2pos rtrj2vel];
rtrj2vel = plamb(rtr2vec,tm2tf,rospos);
rtr2vec = [rtrj2pos rtrj2vel];
[ret re] = ode45('cow',[0 tm2tf],rtr2vec,options);
[rea, reb] = size(re);
rospos = [re(rea,1) re(rea,2) re(rea,3)];
rosvel = [re(rea,4) re(rea,5) re(rea,6)];
vc2 = rtrj1vel - endvel;
vc2m = sqrt(dot(vc2,vc2));
pv2 = vc2/vc2m;
vc3 = rtrj2vel - midc1vel;
vc3m = sqrt(dot(vc3,vc3));
pv3 = vc3/vc3m;
vc4 = stationvel - rosvel;
vc4m = sqrt(dot(vc4,vc4));
pv4 = vc4/vc4m;
total2 = dv1m + vc2m + vc3m + vc4m
%pv2di = pvinit(to2tm, rtrj1pos, rtrj1vel, vc2, vc3);
%[pvec2 pvect2] = pvhist(to2tm, rtrj1pos, rtrj1vel, pv2,pv2di);
%pv3di = pvinit(tm2tf, rtrj2pos, rtrj2vel, vc3, vc4);
%[pvec3 pvect3] = pvhist(tm2tf, rtrj2pos, rtrj2vel, pv3,pv3di);
%plot(pvect2,pvec2)
%hold on
%plot(pvect3+to2tm,pvec3)
%keyboard

% Initial Coast
cstep = 70;
time1 = to2tm + cstep;
time2 = tm2tf - cstep;
[cvect cvec] = ode45('cow',[0 time1],rtr1vec,options);
[cva,cvb] = size(cvec);
coastpos = [cvec(cva,1) cvec(cva,2) cvec(cva,3)];
coastvel = [cvec(cva,4) cvec(cva,5) cvec(cva,6)];
coastvec = [coastpos coastvel];
[rtrj2pos,rtrj2vel,rospos,rosvel] = piter(coastpos,stationpos,time2);
rtr2vec = [rtrj2pos rtrj2vel];
rtrj2vel = plamb(rtr2vec,time2,rospos);
rtr2vec = [rtrj2pos rtrj2vel];
[ret re] = ode45('cow',[0 time2],rtr2vec,options);
[rea, reb] = size(re);
rospos = [re(rea,1) re(rea,2) re(rea,3)];
rosvel = [re(rea,4) re(rea,5) re(rea,6)];
vc2 = rtrj1vel - endvel;
vc2m = sqrt(dot(vc2,vc2));
pv2 = vc2/vc2m;
vc3 = rtrj2vel - coastvel;
vc3m = sqrt(dot(vc3,vc3));
```

```
pv3 = vc3/vc3m;
vc4 = stationvel - rosvel;
vc4m = sqrt(dot(vc4,vc4));
pv4 = vc4/vc4m;
total2 = dv1m + vc2m + vc3m + vc4m
pv2di = pvinit(time1, rtrj1pos, rtrj1vel, vc2, vc3);
[pvec2 pvect2] = pvhist(time1, rtrj1pos, rtrj1vel, pv2,pv2di);
pv3di = pvinit(time2, rtrj2pos, rtrj2vel, vc3, vc4);
[pvec3 pvect3] = pvhist(time2, rtrj2pos, rtrj2vel, pv3,pv3di);
plot(pvect2,pvec2)
hold on
plot(pvect3+time1,pvec3)
keyboard

% Optimize Cusp
vec = [rtrj1pos rtrj1vel 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1];
[t,x] = ode45('j2stm',[0 to2tm],vec,options);
stmim = getstm(x);
for k = 1:1:3;
   for m = 1:1:3;
      p11im(k,m) = stmim(k,m);
      p12im(k,m) = stmim(k,3+m);
      p21im(k,m) = stmim(k+3,m);
      p22im(k,m) = stmim(k+3,m+3);
   end
end
vec = [rtrj2pos rtrj2vel 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1];
[t,x] = ode45('j2stm',[0 tm2tf],vec,options);
stmmf = getstm(x);
for k = 1:1:3;
   for m = 1:1:3;
      p11mf(k,m) = stmmf(k,m);
      p12mf(k,m) = stmmf(k,3+m);
      p21mf(k,m) = stmmf(k+3,m);
      p22mf(k,m) = stmmf(k+3,m+3);
   end
end
pdminus = p21im*pv2' + p22im*pv2di;
pdplus = inv(p12mf)*(pv4'-p11mf*pv3');
djdr = pdplus - pdminus;
djdt = -(pdplus'*rtrj2vel' - pdminus'*midc1vel');
B = -eye(4);
mpos = [aipos(1) aipos(2) aipos(3)];
mtime = to2tm;
g = [djdr;djdt];
tdvc = 1;
while abs(tdvc) > 1.0e-13
d = inv(B)*(-g);
p = d;
delp = [p(1) p(2) p(3)];
delt = p(4);
mpos = mpos + delp;
mtime = mtime + delt;
mtime2 = rost - mtime;
[rtrj1pos,rtrj1vel,midc1pos,midc1vel] = piter(endpos,mpos,mtime);
rtr1vec = [rtrj1pos rtrj1vel];
rtrj1vel = plamb(rtr1vec,mtime,midc1pos);
rtr1vec = [rtrj1pos rtrj1vel];
[ret re] = ode45('cow',[0 mtime],rtr1vec,options);
[rea, reb] = size(re);
midc1pos = [re(rea,1) re(rea,2) re(rea,3)];
midc1vel = [re(rea,4) re(rea,5) re(rea,6)];
[rtrj2pos,rtrj2vel,rospos,rosvel] = piter(mpos,stationpos,mtime2);
rtr2vec = [rtrj2pos rtrj2vel];
rtrj2vel = plamb(rtr2vec,mtime2,rospos);
rtr2vec = [rtrj2pos rtrj2vel];
[ret re] = ode45('cow',[0 mtime2],rtr2vec,options);
[rea, reb] = size(re);
rospos = [re(rea,1) re(rea,2) re(rea,3)];
rosvel = [re(rea,4) re(rea,5) re(rea,6)];
vc2 = rtrj1vel - endvel;
vc2m = sqrt(dot(vc2,vc2));
p2 = vc2/vc2m;
vc3 = rtrj2vel - midc1vel;
vc3m = sqrt(dot(vc3,vc3));
p3 = vc3/vc3m;
vc4 = stationvel - rosvel;
vc4m = sqrt(dot(vc4,vc4));
p4 = vc4/vc4m;
total3 = dv1m + vc2m + vc3m + vc4m
p2di = pvinit(mtime, rtrj1pos, rtrj1vel, vc2, vc3);
p3di = pvinit(mtime2, rtrj2pos, rtrj2vel, vc3, vc4);
vec = [rtrj1pos rtrj1vel 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1];
[t,x] = ode45('j2stm',[0 mtime],vec,options);
sm1 = getstm(x);
for k = 1:1:3;
   for m = 1:1:3;
      p11m1(k,m) = sm1(k,m);
      p12m1(k,m) = sm1(k,3+m);
      p21m1(k,m) = sm1(k+3,m);
```

```
      p22m1(k,m) = sm1(k+3,m+3);
   end
end
vec = [rtrj2pos rtrj2vel 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1];
[t,x] = ode45('j2stm',[0 mtime2],vec,options);
sm2 = getstm(x);
for k = 1:1:3;
   for m = 1:1:3;
      p11m2(k,m) = sm2(k,m);
      p12m2(k,m) = sm2(k,3+m);
      p21m2(k,m) = sm2(k+3,m);
      p22m2(k,m) = sm2(k+3,m+3);
   end
end
pdminus = p21m1*p2' + p22m1*p2di;
pdplus = inv(p12m2)*(p4'-p11m2*p3');
djdr = pdplus - pdminus;
djdt = -(pdplus'*rtrj2vel' - pdminus'*midclvel');
grad = [djdr;djdt];
q = grad - q;
B = B + ((q-B*p)*((q-B*p)'))/(p'*(q-B*p));
q = grad;
tdvc = total2 - total3
total2 = total3;
end
[pvec2 pvect2] = pvhist(mtime, rtrj1pos, rtrj1vel, p2,p2di);
[pvec3 pvect3] = pvhist(mtime2, rtrj2pos, rtrj2vel, p3,p3di);
total3 = total3
plot(pvect2,pvec2)
hold on
plot(pvect3+mtime,pvec3)
keyboard
```

```
% twopmans.m
% Calculate two-impulse evasive maneuvers

% Read in the desired maneuver time
fprintf('\nHow long before the minimum separation point');
fprintf('\nwould you like to perform the avoidance maneuver.\n\n');
fprintf('Enter the [hours minutes secs], include the brackets and spaces.\n');
fprintf('i.e.  2 hours, 30 mins, and 30 secs = [2 30 30]\n');

prior = input('==>  ');
secbef = prior(1)* 3600 + prior(2)* 60 + prior(3);

timespan = [0 coltime-secbef];

% Determin the position of thee spacecraft and the debris at the desired maneuver time
jdate = sdate;
[sbackt, sbackv] = ode45('cow',timespan,scorbit,options);
[sba, sbb] = size(sbackv);
sstartpos = [sbackv(sba,1) sbackv(sba,2) sbackv(sba,3)];
sstartvel = [sbackv(sba,4) sbackv(sba,5) sbackv(sba,6)];
sstartvec = [sstartpos sstartvel];

[dbackt, dbackv] = ode45('cow',timespan,dborbit,options);
[dba, dbb] = size(dbackv);
dstartpos = [dbackv(dba,1) dbackv(dba,2) dbackv(dba,3)];
dstartvel = [dbackv(dba,4) dbackv(dba,5) dbackv(dba,6)];

dbcolpos = [dbcolvec(1) dbcolvec(2) dbcolvec(3)];

% Calculate the ECI avoidance circle locations and the delta v to maneuver to these locations
% and return to the original spacecraft orbit
n = 1;
for i = 0:5:360
   ta = atan2(dbcolvec(2),dbcolvec(1));
   x = threat * cos(i*pi/180)* cos(ta);
   y = threat * cos(i*pi/180)* sin(ta);
   z = threat * sin(i*pi/180);
   threatvec = [x y z];
   avdvec = dbcolpos + threatvec;
   [pmstartpos, pmstartvel, endpos, endvel] = piter(sstartpos,avdvec,secbef);
   pmstartvec = [pmstartpos pmstartvel];
   pmstartvel = plamb(pmstartvec,secbef,endpos);
   pmstartvec = [pmstartpos pmstartvel];
   classical = eci2orbl(pmstartpos,pmstartvel);
   semi = classical(1);
   period = (2*pi*(semi^(3/2)))/sqrt(mu);
   timespan = [0 (period-(period/10))];
   jdate = sdate + ((coltime-secbef)/86400);
   [aroundt,around] = rk4fixed('cow',timespan,pmstartvec,200,0);
   [ara, arb] = size(around);
   aroundpos = [around(ara,1) around(ara,2) around(ara,3)];
   aroundvel = [around(ara,4) around(ara,5) around(ara,6)];
   aroundvec = [aroundpos aroundvel];
   init = period-(period/10);
   jdate = jdate + ((period-period/10)/86400);
```

```
   timespan = [0 (period/5)];
   [aroundt,around] = rk4fixed('cow',timespan,aroundvec,1000,0);
   [ara, arb] = size(around);
   for p = 1:ara
      arpos = [around(p,1) around(p,2) around(p,3)];
      delp = arpos - sstartpos;
      delpm(p) = sqrt(dot(delp,delp));
   end
   [val pos] = min(delpm);
   time = aroundt(pos);
   aroundpos = [around(pos,1) around(pos,2) around(pos,3)];
   aroundvel = [around(pos,4) around(pos,5) around(pos,6)];
   aroundvec = [aroundpos aroundvel];
   jdate = sdate + ((coltime-secbef)/86400);
   timespan = [0 init+time];
   [originalt, original] = ode45('cow',timespan,sstartvec,options);
   [ora, orb] = size(original);
   originalpos = [original(ora,1) original(ora,2) original(ora,3)];
   posdiff = sqrt(dot((aroundpos - originalpos),(aroundpos - originalpos)));
   posdif(n) = posdiff;
   angdiff = rtd*acos(dot(aroundpos,originalpos)/(sqrt(dot(aroundpos,aroundpos))*sqrt(dot(originalpos,originalpos))));
   angdif(n) = angdiff;
   dv1 = pmstartvel - sstartvel;
   dv1m = sqrt(dot(dv1,dv1));
   dv2 =  sstartvel - aroundvel;
   dv2m = sqrt(dot(dv2,dv2));
   totdv(n) = 1000*(dv1m + dv2m);
   tfa(n) = i;
   n = n + 1;
   i
end

% Plot the results
subplot(2,1,1), plot(tfa,totdv);
title('Total Delta V Required for a Two-Impulse Return to Orbit Maneuver')
xlabel('Avoidance Circle Location (deg)')
ylabel('Delta V (m/sec)')

subplot(2,1,2), plot(tfa,angdif);
title('Angular Separation Between the Original Orbit Station and the New Orbit Station')
xlabel('Avoidance Circle Location (deg)')
ylabel('Angular Separartion (deg)')
```